Object-Relational Impedance Mismatch: A Framework Based Approach

Thesis

THE OPEN UNIVERSITY
Department of Computing

OBJECT-RELATIONAL IMPEDANCE MISMATCH:
A FRAMEWORK BASED APPROACH

---

by
CHRISTOPHER JON IRELAND
BA (Hons), MBA (Open)

September 2011

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF THE OPEN UNIVERSITY FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DATE OF SUBMISSION: 29 SEPTEMBER 2011
DATE OF AWARD: 27 MAY 2012

For my wife Loraine

Acknowledgements

I would like to thank my supervisors, David Bowers, Mike Newton and Kevin Waugh for all their support over the years. I would like to thank Marian Petre and all those at the Centre for Research in Computing, who have provided help and support whenever I needed it.

I would also like to extend my gratitude to Aidan O'Malley and Matthew Ruston at Bank Of America Merrill Lynch for their support, my parents June and John Ireland and friends and work colleagues Niven Tasker, Matthew Wall, Christopher Hood and Richard Alfred-Jones who all provided feedback on draft versions of this dissertation.

Finally, I would like to thank my wife Loraine, who has always been there for me.

Abstract

The term impedance mismatch was first used in 1984 to label problems that arise when a program uses a relational database for storage. For example, when transferring data from a relational database into a program any relational data structure is lost because a program operates at the row level. Consequently that data structure must somehow be reproduced when data is returned to a database. There are many such mismatches that cost time and effort to address. As new programming and database languages are introduced other kinds of impedance mismatch are anticipated.

Traditional approaches are concerned with pragmatic solutions to specific problems of implementation. They do not address the underlying cause and offer little rationale for the claim to a "solution". The motivation for this dissertation is to understand the cause of these mismatches so it is then possible to address each of them in an appropriate way.

Problem themes are introduced as a way to make sense of impedance mismatch. Such problems are not independent. Relationships between problem themes demonstrate the complex nature of impedance mismatch and they are used to identify three problems of particular significance. A structure to existing characterisations of impedance mismatch is identified and developed in order to organise the characterisations in a meaningful and useful way. This structure, based on four levels of abstraction, forms the foundation for a new framework.

The framework recognises a separation of concerns between a program and a database across levels of abstraction. At each level is observed a particular kind of impedance mismatch. Through a *dialogue*[1] about a correspondence at each level it is possible to understand and address each kind of mismatch in a structured and consistent way. A technique based on *equivalence* is introduced in support of a dialogue.

The validity of the framework is demonstrated by identifying the cause of some significant mismatches. Across all the levels of the framework are explored both the cause of each *mismatch* and the effect of a solution. A four-stage process is described in support of an exploration and to inform others in the use of the framework. An option for change is linked to a conceptual problem not one of implementation and the fidelity and integrity of an existing solution is improved in a way that can be generalised for other solutions. New insights are also provided into the consequences of one solution.

Understanding cause and effect in this level of detail is not available using an alternative framework described in the literature. However despite the improved understanding of an impedance mismatch and the consequences of a solution there is a limit to what can be achieved using the framework.

---

[1] Italics are used to indicate the first use of a term defined in the glossary.

Related Publications

The majority of the work in this dissertation has been published at conferences and in a journal. Whilst these papers are co-authored with supervisors each paper represents the ideas and research of Christopher Jon Ireland. These ideas include the problem themes, the four-level framework, the four-stage process, the concept of equivalence and an equivalence diagram, the reference silo and the three different kinds of relational database schema possible using SQL:1999. The ideas have been refined, as they have been applied, as a result of numerous discussions with supervisors and conference delegates, and in response to other feedback in the form of review comments from submissions to a number of conferences.

Chapters 3 and 4 are based on the following conference paper.

> Ireland, C., Bowers, D., Newton, M., Waugh, K.: A Classification of Object-Relational Impedance Mismatch. In: Chen, Q., Cuzzocrea, A., Hara, T., Hunt, E., Popescu, M. (eds.): The First International Conference on Advances in Databases, Knowledge and Data Applications, Vol. 1. IEEE Computer Society, Cancun, Mexico (2009) p36-43

Chapter 5 is based on the following conference paper.

> Ireland, C., Bowers, D., Newton, M., Waugh, K.: Exploring the Essence of an Object-Relational Impedance Mismatch - A novel technique based on Equivalence in the context of a Framework. In: Chen, Q., Cuzzocrea, A., Hara, T., Hunt, E., Popescu, M. (eds.): The Third International Conference on Advances in Databases, Knowledge and Data Applications, Vol. 1. IEEE Computer Society, St. Maarten, The Netherlands Antilles (2011) p65-70

Chapter 6 is based on the following journal paper.

> Ireland, C., Bowers, D., Newton, M., Waugh, K.: Understanding Object-Relational Mapping: A Framework Based Approach. International Journal On Advances in Software 2 (2009)

Chapter 7 is based on the following conference paper.

> Ireland, C., Bowers, D., Newton, M., Waugh, K.: Exploring the use of Mixed Abstractions in SQL:1999 - A Framework Based Approach. In: Chen, Q., Cuzzocrea, A., Hara, T., Hunt, E., Popescu, M. (eds.): The Second International Conference on Advances in Databases, Knowledge and Data Applications, Vol. 1. IEEE Computer Society, Les Menuires, France (2010) p22-27

Table of Contents

List of Figures

List of Tables

List of Acronyms

COBOL	Common Business Oriented Language
ISBL	Information Systems Base Language
JDBC	Java Database Connectivity
LINQ	Language Integrated Query
MDA	Model Driven Architecture
NNF	Nested Normal Form
ODBC	Open Database Connectivity
SOID	Synthetic Object Identity
SQL	Structured Query Language
SUDT	Structured User Defined Type
UML	Unified Modelling Language

Chapter 1 Introduction

1.1 Research Context

Over the past thirty years, technologies based on the concepts of an object and a relation have proved useful in the design and development of computer software systems. The concept of an object is used in the design of an object-oriented program, whilst the concept of a relation is used in the design of a relational database. The popularity of each technology means that invariably they are used together.

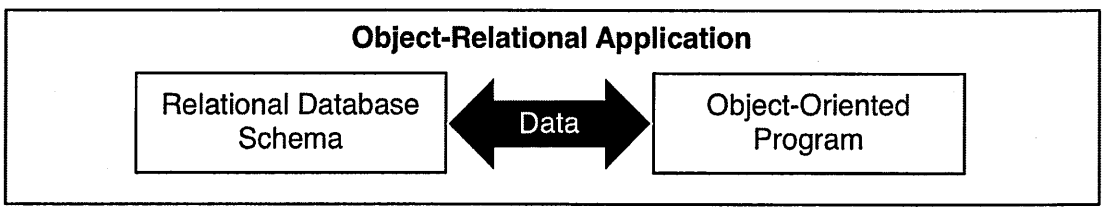


Figure 1 - The Research Context

A computer software system that combines technologies based on the concept of an object, in an object-oriented program, and the concept of a relation, in a relational database, is referred to as an *object-relational application.* On Figure 1 the context of an object-relational application is shown as a box surrounding two further boxes. Those two boxes represent, respectively, a *relational database schema* and the source-code of an object-oriented program. The term *schema* is used to refer both to the collection of classes in an object-oriented program that have a requirement for persistence, and to the description of a relational database.

An object-relational application combines object and relational artefacts because they are suited to a particular role in that application (Russell 2008), p16, but such artefacts are different. Different people with different software development roles, priorities and objectives, making different choices can produce each schema at different times, using different concepts, different technologies, and different languages in different ways, producing different abstractions with different semantics. Each difference contributes in some way to a mismatch.

The impedance mismatch problem occurs because of a number of differences between a program and a database. An *object-relational impedance mismatch* occurs because of differences between an object-oriented program and a relational database. Each such difference is referred to as a mismatch.

If an object-oriented program uses a relational database for storage, data must pass between the two and typically in both directions. Such a flow is depicted on Figure 1 as a black arrow labelled "Data". During the development of an object-relational application issues and tensions arise because of mismatches. Each mismatch interferes with the seamless transfer of data between an object-oriented program and a relational database and takes time and effort to address (Keller, Jensen et al. 1993).

Neward (Neward 2007) observes that the use of a single language for the schema of a program and the schema of a database would avoid some of the problems of impedance mismatch. He makes this observation for an object-oriented program that

uses an object-oriented database for storage. However such an application based on a single set of concepts is not within the scope of this dissertation because it is not an object-relational application.

1.2 Research Justification

The different technologies for implementing a solution to a mismatch (for example Hibernate (Bauer and King 2007) and Oracle TopLink (TopLink)), and the existence of guidelines (Marguerie 2007) and metrics (Holder, Buchan et al. 2008) for selecting a solution suggest that problems of an object-relational impedance mismatch are not uncommon or trivial. However in spite of the popularity of combining these technologies there is not a single and coherent understanding of what constitutes impedance mismatch.

1.2.1 Making Sense of Impedance Mismatch

Copeland & Maier (Copeland and Maier 1984), Ambler (Ambler 2003) and Neward (Neward 2006) characterise the problem of impedance mismatch in different ways. Each takes a different perspective and uses a different language to describe the problem.

In all three characterisations a relational database is used for storage. However Copeland & Maier are concerned with issues involving a procedural language whilst Neward and Ambler consider issues involving an object-oriented language.

Neward (Neward 2006) makes explicit a number of concerns, problems and issues faced by those developing an object-relational application. He characterises the problem of object-relational impedance mismatch as a "quagmire of issues" for which he provides a number of examples including the "partial object problem" and the "load-time paradox".

Ambler talks about cultural and technical difficulties in the design of an object-relational application. A technical difficulty is concerned with a difference of technology. A cultural difficulty is concerned with a difference of agenda between two communities, one responsible for the object-oriented program and the other responsible for the relational database.

Copeland & Maier characterise both a difference of data type, as one difference between two programming languages, and a difference of *paradigm*. A paradigm is a particular way of understanding a world. A conceptual framework embodies a particular paradigm and determines the building blocks of a language. In an object-relational application there are two conceptual frameworks: one based on the concept of an object, the other based on the concept of a relation.

Whilst Copeland & Maier, Ambler and Neward each describe impedance mismatch from a particular perspective it is not clear whether each description relates to the cause of a mismatch or the symptoms. A way to organise the different characterisations of the problem will help to understand impedance mismatch. It will

then be possible to explore the answer to questions such as: does a mismatch of data type result from an issue of language or the concepts on which a language is based? and, does a mismatch occur because of a choice of design, a difference of language or a difficulty of culture?

There are other kinds of impedance mismatch (Lammel and Meijer 2006),p194 and it is likely that there will be more in the future. As programming languages such as Ruby (Ruby 2011) and Python (Python 2011) and database technologies such as NoSQL (Couchbase 2011) are combined problems of impedance mismatch can be expected. If there is a way to understand the cause of object-relational impedance mismatch then that might also be used to understand the cause of other kinds of impedance mismatch.

1.2.2 Understanding the Cause of a Mismatch

In this dissertation the received wisdom that the problem of object-relational impedance mismatch has been solved is brought into question. In the literature and in practice, there are many solutions to the problem so a choice of solution must be made. Typically a solution implements a correspondence between those classes in the design of an object-oriented program that have a requirement for persistence, and the schema of a relational database. Each solution is referred to as an object-relational mapping strategy (or *mapping strategy*).

An *appropriate solution* will address the cause of a mismatch. A correspondence may be false, or at best incomplete, because it does not address the cause of a mismatch. A mapping strategy may therefore involve a compromise if it only deals with the symptoms of a problem. Such a mapping strategy is referred to as an *acceptable solution.* If the cause of a mismatch is understood then it will be possible to understand the consequences of a choice of solution and make an informed decision about a mapping strategy.

In the literature there is little by way of discussion of the causes of a mismatch. If the cause of a mismatch is not understood then those responsible for a mapping strategy will not know whether it is appropriate. However the availability of a mapping strategy reinforces the belief that the problem of an object-relational impedance mismatch has been solved.

Object-relational impedance mismatch is not a single well-defined problem. It is a complex problem that involves many mismatches. For each mismatch there may be a choice of mapping strategy. Because an acceptable solution addresses the symptoms of a mismatch rather than the cause, it may not help understand the cause of a mismatch.

An acceptable solution will involve a compromise because, in this case, the real cause of a mismatch is not between the schema of an object-oriented program and the schema of a relational database. For example a class and a table are different

abstractions and so the use of a table to represent the structure of a class, typical of many mapping strategies, can lead to problems. The cause of this mismatch might be a problem of language or a problem of paradigm. Without a way to understand the cause of a mismatch it cannot be known where that mismatch should appropriately be addressed.

Neward (Neward 2006) concedes that "so long as programmers prefer to use object-oriented programming languages to access relational data stores, there will always be some kind of object-relational mapping taking place -- the two models are simply too different to bridge silently". Until the cause of the various mismatches involved is understood it will not be known whether a more effective "bridge" can be built, in the right place, involving less compromise.

It is important to address a mismatch in an appropriate way. Keller (Keller, Jensen et al. 1993) claims that between twenty and thirty percent of code in an object-relational application is concerned with mismatches. Neward (Neward 2006) labelled such mismatches "the Vietnam of Computer Science" because an initial quick win based on the received wisdom is rapidly replaced by a quagmire of issues. Ambler (Ambler 2003),p106 refers to deceptive similarities between an object-oriented program and a relational database and subtle differences that lead to difficulties. Brown (Brown and Whitenack 1994) uses the term "chasm" to describe a mismatch. An appropriate solution will avoid the quagmire and the effort in dealing with symptoms.

An understanding of the cause of a mismatch can inform a choice of mapping strategy. A tool such as Hibernate (Bauer and King 2007) embodies a number of mapping strategies from which a programmer must make a choice. Whilst such a tool implements mapping strategies, those responsible for the design and implementation of an object-relational application might wish to understand the consequences of a choice for reasons such as those of performance (Zyl, G. Kourie et al. 2006). Ambler (Ambler 2003), p223 provides practical guidance in support of such a choice, but that choice will be ill informed if an appropriate solution is not first understood.

Understanding the cause of a mismatch and the consequences of a solution will help to avoid a mismatch in the future. An understanding of the cause of a mismatch may eventually lead to a new or improved way to design an object-relational application, a new or improved way to make a choice of a mapping strategy, or a new or improved mapping strategy. It may also lead to a fundamental change such as a change to the way the concepts involved are understood, or a change to a language used to implement an object-relational application.

1.3 Research Question

The research question is: can a more encompassing perspective on object-relational impedance mismatch be developed, one that provides actionable insights into the cause of a mismatch?

The hypothesis explored in this dissertation is that a general framework based on a synthesis of both the theory and practice of impedance mismatch provides a foundation for understanding the cause of a mismatch.

Exploring the hypothesis leads to a number of sub-questions including:

1. In what way does the framework reconcile the different perspectives in the literature?
2. Can the framework be used to drive an analysis process, and if so, how might that process be structured?
3. Does considering the cause of a mismatch provide insights into a solution?

1.4 Research Approach

In outline, the research approach is concerned with:

1. Making sense of impedance mismatch;
2. Identifying important aspects of the problem;
3. Developing a framework based on a synthesis of theory and practice;
4. Developing a process to provide guidance in the use of the framework; and
5. Validating that framework as a way to identify the cause of a mismatch.

In order to make sense of impedance mismatch common themes are identified amongst descriptions of the problem in both the theory and practice. Between themes relationships are explored as a way both to understand the nature of impedance mismatch and to identify problems of particular importance.

The characterisations of Copeland & Maier, Ambler and Neward each present the problem of impedance mismatch in terms of a particular abstraction. Copeland & Maier focus on issues of concept and language. Neward and Ambler characterise pragmatic solutions to problems in the design of an object-relational application. These characterisations are synthesised as a framework in which each characterisation represents a different level of abstraction over an object-relational application. All the levels of the framework can be used to explore the cause of a mismatch.

Work on the framework has been presented both at conferences and in a peer-reviewed journal. In this dissertation the framework is validated in three ways. First the framework is used to understand the cause of three significant mismatches and so demonstrate the efficacy of the framework; second the concerns of the framework are compared with those of another framework described by Fussell (Fussell 1997) and so demonstrate the novel perspective of the framework; and third the consequence of a change to a language (Eisenberg and Melton 1999) as one solution to a mismatch are explored and so demonstrate an emergent property of the framework.

1.5 Dissertation Structure

Chapter 1 presented the rationale and justification for a change in how the problem of object-relational impedance mismatch is understood and approached. Chapter 2 presents a more detailed characterisation of the problem and the ways in which it is currently addressed. Chapter 3 presents an understanding of the problem of object-

relational impedance mismatch and identifies three problems of particular importance.

Chapter 4 introduces the proposed framework. The framework is compared with that of Fussell (Fussell 1997) in order to illustrate the distinctive nature of the framework, which shifts the focus from issues of implementation to the cause of a mismatch. The framework changes thinking about both a mismatch and a mapping strategy. An analysis of a mapping strategy is provided to illustrate the identification of the cause of a mismatch.

Chapter 5 introduces the notion of "equivalence", used in this dissertation to mean the preservation of semantics between the schema of an object-oriented program and the schema of a relational database. Chapter 5 also introduces an equivalence diagram, which is used to explore the basis of a correspondence between the schema of an object oriented program and the schema of a relational database, and hence to understand a mapping strategy.

Chapter 6 introduces and demonstrates a four-stage process that provides guidance in the use of the framework. In the context of the process the framework is used, in a systematic way, to understand the cause of a mismatch and to suggest solutions.

Chapter 7 demonstrates how the framework is used to explore the consequences of a possible solution to object-relational impedance mismatch at the language level. The

conclusions are presented in Chapter 8 and the opportunities for future work are described in Chapter 9.

Chapter 2 Literature Review

2.1 Introduction

This chapter sets out to understand impedance mismatch. There are various characterisations of the problem in the literature and in practice. Each characterisation is based on a particular perspective. Relationships between these characterisations are explored in order to illuminate the concern of each.

In practice, the symptoms of a mismatch materialise as a difference between an object-oriented program and a relational database. A generalised model of the process of object-relational application development is developed and used as a structure to understand some of the differences and their consequences for a mismatch.

Solutions to the problem of object-relational impedance mismatch are surveyed. Each solution approaches the problem in a different way. The concern of each solution for understanding the cause of a mismatch is examined.

2.2 Impedance Mismatch

The term "impedance mismatch" was first used in 1984 by Copeland & Maier (Copeland and Maier 1984) to refer to problems that occur when a software system is implemented using two different languages. They distinguish specifically a programming language from a database language.

Copeland & Maier were the first to label and characterise impedance mismatch but they do not provide a sound basis for an understanding of the cause of a mismatch. However their work is still cited in the literature (for example (Meijer and Bierman 2011)) when there is a concern for issues of impedance mismatch.

The objective of Copeland & Maier was to introduce the GemStone (Copeland and Maier 1984) object-oriented database. They describe "shortcomings with commercial database systems" but then devote only a paragraph to the subject of an impedance mismatch. They describe two mismatches: conceptual and structural. A conceptual mismatch they define as a difference of programming paradigm. A structural mismatch they define as a difference of data type.

Whilst the term paradigm was originally intended to describe the set of practices that define a scientific discipline at any particular period of time (Kuhn 1970), it has been used in computing as a label for a particular viewpoint. A conceptual framework embodies a particular paradigm and determines the building blocks of a language.

Copeland & Maier consider the consequences of a mismatch of structure and describe one symptom, that a data structure is "reflected back" at the interface between a COBOL (COBOL) program and a relational database. By reflected they mean that a data structure of a database is not reproduced in a COBOL program. They explain that such reflection occurs because the languages concerned do not support the same data

types. They do not explain why there is a difference of data type but conclude that GemStone will use a single language in order to avoid this problem.

Copeland & Maier do not consider whether a difference of programming paradigm and a difference of language are related. However the data structure and the processing structure of a language do reflect a particular paradigm. COBOL is labelled a "procedural" language because of the nature of its processing structure. Languages such as Java (Weber 1997) and C++ (Stroustrup 1997) are often referred to as "object-oriented" because they employ the concept of an object as their data and processing structure. A programming paradigm therefore has an influence on the artefacts employed in a programming language so a difference of data type may be a consequence of a difference of programming paradigm.

Ambler (Ambler 2003), p113 presents a definition of object-relational impedance mismatch as technical and cultural difficulties that occur during the development of an object-relational application. A technical difficulty is concerned with a mismatch of technology. A cultural difficulty is concerned with the agenda of two communities, one responsible for an object-oriented program and the other responsible for a relational database.

Ambler explains a number of issues that result from the problem of object-relational impedance mismatch but he does not make clear how his work relates to that of Copeland & Maier. It is not clear for example whether the structural and the

conceptual mismatches described by Copeland & Maier are two kinds of a technical difficulty, or whether they refer to a different concern. Because a paradigm underpins the building blocks of a language, both a structural mismatch and a conceptual mismatch could be categorised as a technical difficulty and a cultural difficulty.

The concern of Ambler is avoiding or reducing the impact of object-relational impedance mismatch. Ambler suggests that object-relational impedance mismatch can be mitigated if 1) the coupling between a program and a database is reduced; 2) a database is designed well; and 3) by keeping a design clean, in other words by making a design easy to understand and to modify. Such concerns are relevant to those who must deal with a mismatch but it does not help to understand the cause.

Neward (Neward 2006) makes explicit a number of concerns, problems and issues of object-relational impedance mismatch. He characterises the problem of object-relational impedance mismatch as a "quagmire of issues" for which he provides a number of examples including the "object-to-table mapping problem" and the "load time paradox". Neward does not make clear whether the list of examples is comprehensive, typical or those most easily recalled.

Ambler (Ambler 2003), p113 suggests that a technical difficulty can be overcome when project team members understand the basics of object and relational technologies. However if object-relational impedance mismatch is a quagmire of issues as Neward suggests, then understanding the basics might not be sufficient.

The characterisation of Neward is a first step towards an understanding of the nature of a mismatch. However a characterisation is not sufficient to understand the cause of a mismatch. Neward does not explore the cause of a mismatch rather he describes the way a mismatch is typically addressed using an analogy based on the Vietnam War: "...a quagmire which starts well, gets more complicated as time passes, and before long entraps its users in a commitment that has no clear demarcation point, no clear win conditions, and no clear exit strategy."

It is not clear how each of Newards characterisations relate to those of Copeland & Maier and Ambler. For example the object-to-table mapping problem could be a symptom of a technical difficulty because an object and a table are artefacts of separate technologies, a symptom of a structural mismatch because an object and a table are separate data structures, or a symptom of different paradigms or cultures because they are the product of choices made by different teams.

Cook (Cook and Ibrahmi 2005) claims to clarify the problem of integrating programming languages and databases. They develop a set of criteria based on three categories: typing, optimisation, and reuse, and use these to evaluate a number of solutions to the problem of impedance mismatch. They acknowledge cultural and technical difficulties but their concern is with a choice of solution rather than the cause of mismatch.

There are different ways to characterise the problem of object-relational impedance mismatch. It is not clear how each characterisation relates to the others, whether a particular characterisation refers to the cause of a mismatch or a symptom, whether the list of characterisations is complete, and why each characterisation was chosen. Whilst Cook claims to clarify a mismatch, their concern is with solutions. Ambler and Neward consider issues beyond those of technology but it is not clear whether Copeland & Maier, Ambler and Neward describe the cause or a symptom of a mismatch.

2.3 A Generalised Model of Software Development

The schema of an object-oriented program and the schema of a relational database together lie at the heart of an object-relational application. The combination of these two schemas provides the context for a mismatch so it is important to understand how a schema is produced. This section sets out a general model of software development, based on the literature, that is then used to understand some of the choices made and their consequences for the form and content of each schema.

The ANSI three-tier database model (Figure 2) provides a structure for understanding the context of a database schema (Brown 2001), p531. Each level of the ANSI model reflects how a particular group of stakeholders view a database. The conceptual level is concerned with a description of a universe of discourse as a conceptual model and so reflects the perspective of an end-user of a database, whilst the logical level is concerned with the schema of a database and so reflects the perspective of those

responsible for the design of a database. The physical level is concerned with the actual organisation of data storage on particular computer hardware. Whilst a schema is one possible view of data the ANSI model is not a process of software development and it does not prescribe how a schema is produced.

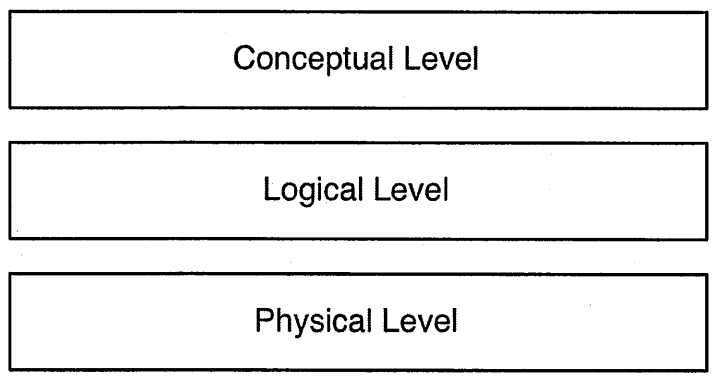


Figure 2 – The ANSI Three-Tier Model

Because they are two parts of the same application the schema of an object-oriented program and the schema of a relational database are a description of the same universe of discourse. A *universe of discourse* is a model of reality containing all concrete or abstract things that are of interest (Griethuysen 1982), pA-6. Typically a schema is not a direct representation of a universe of discourse rather a schema is the product of a process that starts with a universe of discourse.

A conceptual model is an essential cornerstone of many information system analysis and design methodologies (Recker and Rosemann 2010). Marcos (Marcos and Cavero 2002) observes that, "one of the most important objectives in conceptual modelling consists in narrowing the gap between the real world and its representation". This

objective has motivated developments in semantic data modelling (Peckham and Maryanski 1988) but the semantics of a model depend on more than the language used to describe that model.

Poels (Poels, Maes et al. 2005) describes metrics for measuring the "perceived semantic quality" of a conceptual model. These metrics relate to such things as the accuracy, relevance and completeness of a model with respect to a universe of discourse. Consequently it is important for the quality of a conceptual model that those who produce a conceptual model are able to describe those aspects of a universe of discourse in a conceptual model in the requisite detail and with the requisite completeness.

A schema is different from a conceptual model because a schema is concerned with artefacts of implementation, that is, a model that can be executed on a computer. Kotiadis (Kotiadis and Robinson 2008), p952 observes that a computer model (a schema) is a software specific design and software representation of a conceptual model. Consequently, for Kotiadis, at some point during the development of an application a conceptual model is transformed into a schema.

It is important that those who design a schema are able to make the intended interpretation of a conceptual model. Poels (Poels, Maes et al. 2005), p384 observe that the quality of a conceptual model relates directly to the accuracy of a task. They make this observation in the context of the task of information retrieval but the

transformation from a conceptual model to a schema is another task that employs a conceptual model. Bowers (Bowers 2003) conclude that it is by no means certain that those who produce a model will produce a good quality model or make the same choice of concept or language for describing that concept. Because two people can have different perspectives and priorities, it is possible to interpret a universe of discourse in different ways. Consequently there can be a number of different schemas, each describing the same conceptual model.

Smith & Smith (Smith and Smith 1982) identify conceptual design, the production of a conceptual model, and a physical design, the production of a schema, as two phases of a process of database design. An (An, Hu et al. 2010) observes that typically the design of a relational database starts with the design of a conceptual model which is then transformed into a relational database schema. They are concerned with maintaining consistency between a conceptual model and a schema in response to a change to either model. They do not describe the transformation from a conceptual model to an implementation model rather they assume that a suitable transformation has already been applied.

A transformation between two models is also part of the design of a program. For example the UML (Rumbaugh, Jacobson et al. 2005) is a language for the description of a model based on the concept of an object. A model described using the UML can be transformed into the source code of an object-oriented program. The design and definition of such transformations form the foundation of the Model Driven

Architecture (MDA) (OMG). However, typically, the design of a program is a separate activity from the design of a database.

Emerging from the work of Smith & Smith, Kotiadis and An is an apparent shift in emphasis. As the development of a schema progresses the emphasis shifts from describing a universe of discourse to representing the semantics of a conceptual model in a schema. Such a distinction between different models is typical of software development: both (Smith and Smith 1982) and (Brown 2001) make the distinction between a conceptual model and a physical model; Hainaut (Hainaut 2006) distinguishes a conceptual model and a logical model; Kotiadis (Kotiadis and Robinson 2008) distinguishes a conceptual model and a computer model. Whilst there are differences in terminology when referring to a schema, software development can be considered a process of model *transformation*. That is, a collection of choices about how best to represent a universe of discourse as a conceptual model and a conceptual model as a schema.

The transformations from a universe of discourse to a conceptual model and from a conceptual model to a schema can involve a loss of semantics because both a conceptual model and a schema are an abstraction: *"An abstraction of some system is a model of that system in which certain details are deliberately omitted. The choice of the details to omit is made by considering both the intended application of the abstraction and also its users. The objective is to allow users to heed details of the system which are relevant to the application and to ignore other details."* Smith (Smith and Smith

1977b). Consequently understanding these transformations is important to understanding the form and content of each schema.

In this dissertation the term *conceptual model* refers to a representation of selected aspects of a universe of discourse without concern for how those aspects may be represented in a computer. The term *implementation model* refers to a representation of a conceptual model in computer software (what Smith & Smith refer to as a physical model and Kotiadis refers to as a computer model). A schema is a form of implementation model.

An implementation model is composed in terms of a particular *implementation language*. An implementation language is a programming language or a language used to describe a database. Typically there is a choice of implementation language for example Java (Weber 1997), COBOL (COBOL), SQL (ISO 2003) and C++ (Stroustrup 1997), and the language used to express an implementation model, such as a program or a database schema, may not be the same as the language used to express a conceptual model.

The distinction between a conceptual and an implementation model is made in order to explore the choices and the products of a transformation and their consequences for the form and content of a schema. When a transformation is applied to a model it produces a description as another model. A generalised process of software

development, involving two distinct models: a conceptual model and an implementation model, is presented in Figure 3.

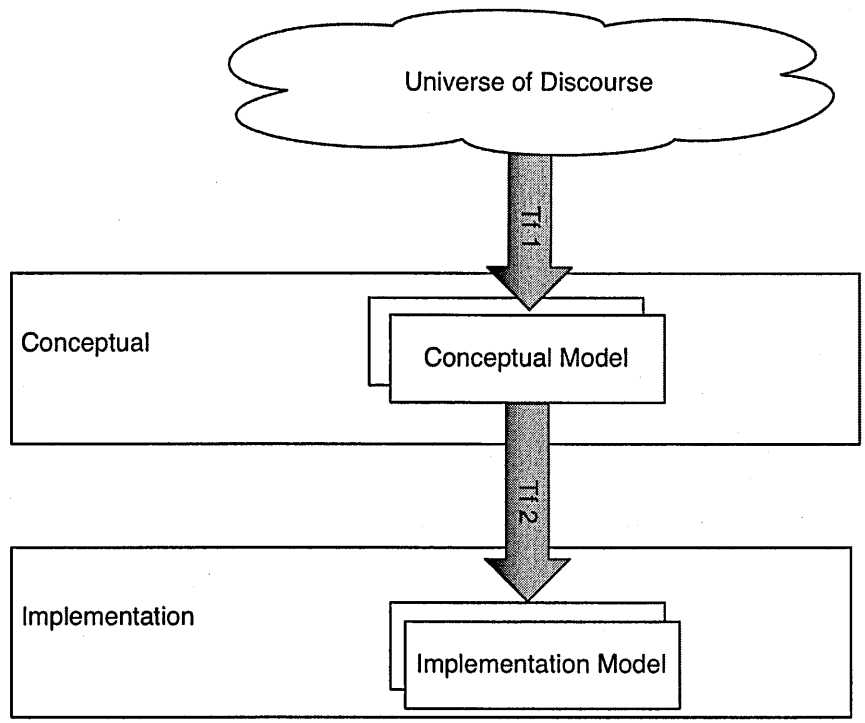


Figure 3 - Software Development as a Process of Model Transformation

The generalised process (Figure 3) results in an implementation model but starts with a universe of discourse. The first transformation (Tf1) yields a single conceptual model of this universe. A second transformation (Tf2) yields an implementation model from a conceptual model. The sequence of transformations Tf1 followed by Tf2 results in an implementation model.

While it is possible to ignore certain details in order to focus on that which is considered important (Smith and Smith 1977b), a consequence of a model is a possible loss of semantics. A transformation such as Tf1 results in a loss of

information about a universe of discourse. It is the job of those who produce a model to make a choice of concept, a choice of language with which to describe a concept, and to decide how to manage those semantics on one model that cannot be represented in another.

This process (Figure 3) provides a structure to explore the development of a schema and prompts questions such as who is responsible for each model; what are they trying to achieve; when transforming from one model to another what choices must they make and why; and what are the consequences of a particular choice? The answer to each question will help to illuminate differences between models and any compromises that are made.

The next section applies this generalised model of software development to understanding the development of the two schemas that comprise an object-relational application. The objective is two fold: first to introduce the models involved and second to understand the choices of transformation that produce each schema and the context for a mismatch.

2.4 Object-Relational Application Development

In practice, an object-relational application provides the context for a mismatch. However it is not clear how the development of an object-relational application should proceed because there are options. The schema of a relational database and the schema of an object-oriented program can be developed independently and from

different conceptual models (Ambler 2003), p261 or from the same conceptual model (Marcos, Vela et al. 2003); the schema of an object-oriented program can be based on the schema of a relational database (Hohenstein 1996) or the schema of a relational database can be based on the schema of an object-oriented program (Keller, Jensen et al. 1993), p527.

Ambler (Ambler 2003), p261 argues that the schema of a relational database should not drive the design of an object-oriented program, nor should the schema of an object-oriented program drive the design of a relational database. In essence, his argument is that design of each schema is influenced by a separate conceptual framework and motivated by separate concerns. Brown (Brown 2001), p622 observes that, in practice, the design of a database schema will progress concurrently with that of an object-oriented program. Consequently the distinction is made, for now, between two separate processes of model transformation. Each process starts from the same universe of discourse but one of the processes results in the schema of an object-oriented program whilst the other process results in the schema of a relational database.

Reflecting the concerns of both Ambler and Brown, the model for the development of an object-relational application involves two separate conceptual models: a conceptual data model and a conceptual object model. Each conceptual model is transformed into a separate implementation model. One implementation model is the schema of a relational database and the other is the schema of an object-oriented

program. Those who develop an object-relational application combine these two schemas. The situation is summarised in Figure 4.

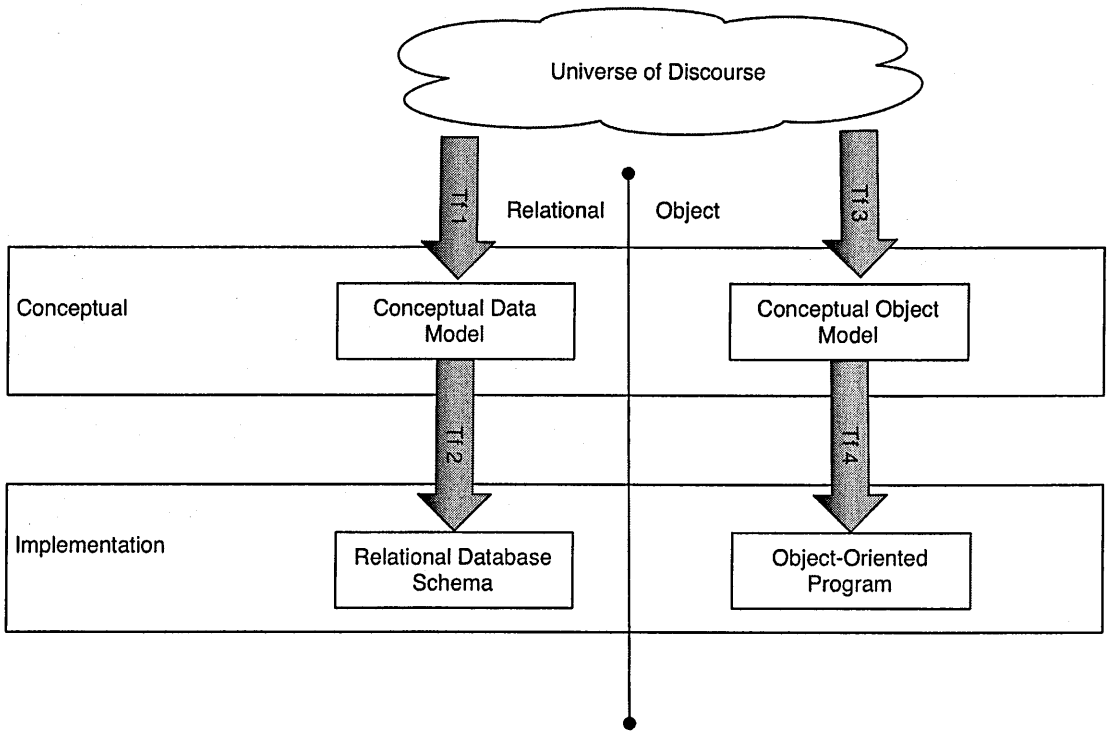


Figure 4 - Model Transformations in Object-Relational Application Development

Transformation Tf1 and transformation Tf3 each produce a distinct conceptual model because of differences between the conceptual frameworks. The object framework, represented by models shown down the right side of Figure 4, is processing centric and requires that a universe of discourse be thought of as a network of interacting objects. The relational framework, represented by models shown down the left side of Figure 4, is data centric and requires that a universe of discourse be thought of as a collection of relations. Starting from the same universe of discourse, the transformation Tf1 results in a conceptual data model whilst transformation Tf3 results in a conceptual object model.

In the next section the model in Figure 4 is used as a structure to illuminate the development of an object-relational application. In addition to the questions raised in section 2.3 using this model it is possible to explore differences of design between conceptual models and between implementation models; the choices that are made in each transformation from a conceptual model to an implementation model, and the separate concerns of those responsible for the design of each schema.

2.4.1 Different Conceptual Models

The semantics of a conceptual data model are important because they are the foundation for the design of a database. Davies (Davies, Green et al. 2004) observes that the E-R model (Chen 1976) is a popular choice for expressing a conceptual model of data. However it was not the objective of Chen (Chen 1976) to provide guidance on the selection of things from a universe of discourse or whether they are represented as an entity-set or as a relationship. It is left to those who produce a model to decide how best to describe a thing of interest from a universe of discourse.

The semantics of a conceptual object model are important because they are the foundation for the design of an object-oriented program. Capretz (Capretz 2003) describes the development of thinking in terms of an object. There is not one single source to which they attribute the idea of an object. The object concept, with varying terminology emerged almost independently in various branches of computer science

during the 1970s, such as system simulation, operating systems, artificial intelligence and data abstraction.

There are different interpretations of an object. In the context of a model the term "object" is used widely in the literature but to refer to different things. An object can be "a thing that exists" (Blaha, Premerlani et al. 1988), something in the "real-world" (Smith and Smith 1977a), (Schmid and Swenson 1975), a representation of something in a universe of discourse (Hall, Owlett et al. 1976), a concept (Zhang and Ritter 2001), "something that has state, behaviour and identity" Booch in (Armstrong 2006), "a discrete entity [sic] with a well-defined boundary and identity that encapsulates state and behaviour" (Rumbaugh, Jacobson et al. 2005), p482, "an instance of a class" (Rumbaugh, Jacobson et al. 2005), p482, "a person, place, thing, event, concept, screen or report" (Ambler 2003), something in a design (Giguette 2006) or a design notation (Rumbaugh, Jacobson et al. 2005). Each interpretation has consequences for the semantics of a conceptual object model. A model can combine two interpretations and it might not be clear to those who use a model which interpretation is being used.

A conceptual data model and a conceptual object model represent the same universe of discourse in a different way, using different modelling languages and different concepts, each reflecting a particular perspective and priority. Consequently, using the model of software development in Figure 4, those who design the schema of an

object-oriented program and those who design the schema of a relational database start from a different conceptual model.

2.4.2 Different Schemas

This section explores transformations from a conceptual model to an implementation model. A conceptual data model is transformed into the schema of a relational database. A conceptual object model is transformed into the schema of an object-oriented program. These transformations are respectively Tf2 and Tf4 on Figure 4. Differences between these two schemas provide the context for object-relational impedance mismatch.

In 1970 Codd (Codd 1970) introduced a model for shared data based on the concept of a relation. This model is referred to as the relational model. The language of the relational model is based on the relational algebra. A relational database is a data store that adheres to the semantics of the relational model. The work of Codd is underpinned by the work of Childs (Childs 1968) and can trace its development back to work on set theory.

Those responsible for the design of a relational database undertake to transform a conceptual data model into a relational database schema. A relational database schema is a description of data (Date 1986), p361 and the rules of that data in the language of a particular database. On Figure 4 the transformation of a conceptual data model to a relational database schema is labelled Tf2.

There are a number of languages for the description of a relational database schema including QUEL (Ingres 2006), ISBL (Todd 1976) and SQL (ISO 2003). SQL became the language of choice for the description of a relational database schema. In 1986 ANSI published a standard specification for SQL, which was then adopted as an international standard in 1987. Since then there have been a number of editions extending the international standard in 1989, 1992, 1999, 2003 and 2008. Because there are interpretations of the SQL standard by vendors such as Oracle and Sybase in this dissertation the SQL standard is used and the version clarified as appropriate by using the year it was introduced. For example when referring to the version of SQL introduced in 1999 the label SQL:1999 is used.

There is a distinction between the relational model and SQL as an interpretation of the relational model. SQL is a language based on the principles of the relational model. However there are differences of semantics between the relational model and SQL. For example a tuple of a relation in the relational model is distinct by definition whereas a row of a table in SQL can be a duplicate of another row.

In this dissertation the term *relational database design* refers to the transformation from a conceptual data model to a relational database schema. The process of relational database design involves a number of choices. Each design choice is based on factors such as the consistency of data, access performance, space used and the availability and security of data (Date 1986), p12-15.

Codd claims the relational model facilitates an efficient representation of data because it "forms a sound basis for treating derivability, redundancy, and consistency of relations" (Codd 1970), p381. To achieve this objective Codd introduces the idea of a normalised form of relation. A normal form is not part of the relational model rather it results from the application of a separate process during the design of a relational database. One objective of the process of normalisation is the removal of redundant data as a first step towards data consistency. Further details of both normal forms and the process of normalisation can be found in (Date 1986), Chapter 17. A consequence of normalisation is that data about something from a universe of discourse can be spread across several relations.

The same concept on a conceptual data model can be represented in different ways in the schema of a relational database. For example, a relationship between two entities on an E-R model (Chen 1976) can be represented as a foreign key or as a table. However the same artefact in the schema of a relational database can also represent two different concepts. For example a table can be used to represent an entity or a relationship between entities (Soutou 2001), p88. Consequently it can be necessary to refer to a corresponding conceptual data model in order to correctly interpret the semantics of a relational database schema.

In this dissertation the term *object-oriented program design* refers to the transformation from a conceptual object model to the schema of an object-oriented program. A programmer undertakes the design of a program. The process of object-

oriented program design is depicted by transformation Tf4 on Figure 4. A programmer decides how a conceptual object model is best described using a particular object-oriented programming language.

Contrary to the received wisdom (Capretz 2003), p7, thinking in terms of an object is not a seamless process from a universe of discourse to the schema of an object-oriented program. The language used for a conceptual object model is not the same language used for the schema of an object-oriented program.

The Unified Modelling Language (UML) (OMG 2004) is a language for the expression of an object model. The UML defines the syntax and semantics of a collection of modelling notations. Typically a class model is used to describe a data structure whilst a sequence diagram provides a view on the interaction between objects. The UML is not an executable programming language although it does contain an object constraint language. Consequently a different language must be used for the schema of an object-oriented program.

The use of the term "object" both in a conceptual object model and an object-oriented program, for some, provides an illusion of continuity (Capretz 2003), p7. The expectation is that a transformation is seamless. However there are differences between a conceptual object model and the schema of an object-oriented program.

Most object-oriented languages can trace their origins back to Simula (Capretz 2003), p4. Today there is a choice of object-oriented programming languages including C++ (Stroustrup 1997), Java (Weber 1997), Smalltalk (Lalonde 1994) and Objective-C (Kochan 2008). Each language has its own syntax and semantics. Because there is a choice of language it is possible to produce different schemas for an object-oriented program from the same conceptual object model. The language C++ supports multiple-inheritance so a class can inherit both a definition of data (attributes) and processing (methods) from more than one parent class. The semantics of a Java class are different in this respect. For example Weber (Weber 1997), p1084 notes that a Java class can inherit the attributes and methods of a single parent class although a class can also implement a number of interfaces.

There are different interpretations of an object. In the context of a program the term "object" is used widely in the literature but to refer to different things. An object can be "something in an object-oriented program" (Weber 1997), a structure in memory that may contain different data over the execution of a program (Wieringa and De Jonge 1991), or a "chunk of storage" (Kent 1991),p3 in an executing program. Over time, and as an object is created and destroyed, a chunk of storage, such as an area of memory, can contain a representation of the same or a different object.

It is by no means certain that the structure of an object-oriented program will be the same as the structure of a conceptual object model. It is possible to transform a concept on a conceptual object model to an implementation model in different ways.

For example it is a choice of implementation whether a composition on a UML class model is represented as two separate classes or as a single class in the schema of an object-oriented program.

2.4.3 Different Cultures

There are differences other than those of language and concept. The design of a conceptual data model and the design of a relational database schema can be split between different roles within a development team, and a single person in that team may not hold both roles. Similarly, responsibility for the design of a conceptual object model and the schema of an object-oriented program can be split. Those responsible for each model will make a number of role specific choices and not necessarily the same ones.

Cultural impedance mismatch (Ambler 2003),p111 recognises the separation of concerns between those responsible for an object-oriented program and those responsible for a relational database. Each person will make a choice of language and concept based on such things as their power, role, experience, education and priority. The schema of an object-oriented program and the schema of a relational database will reflect each choice. Consequently impedance mismatch is a problem of different cultures as well as a technical problem.

2.4.4 The Consequences of Two Different Schemas

The development of an object-relational application can be thought of as two separate processes, the products of which are the schema of an object-oriented program and

the schema of a relational database. Thinking about the development of an object-relational application in this way provides a coherent structure for understanding both theory and practice. This section sets out the consequences of two separate schemas.

Each process involves different models expressed in different languages, each reflecting different choices made by people occupying different roles and reflecting different priorities and concerns. The result is the schema of an object-oriented program and the schema of a relational database that each describes the same universe of discourse but in a different way.

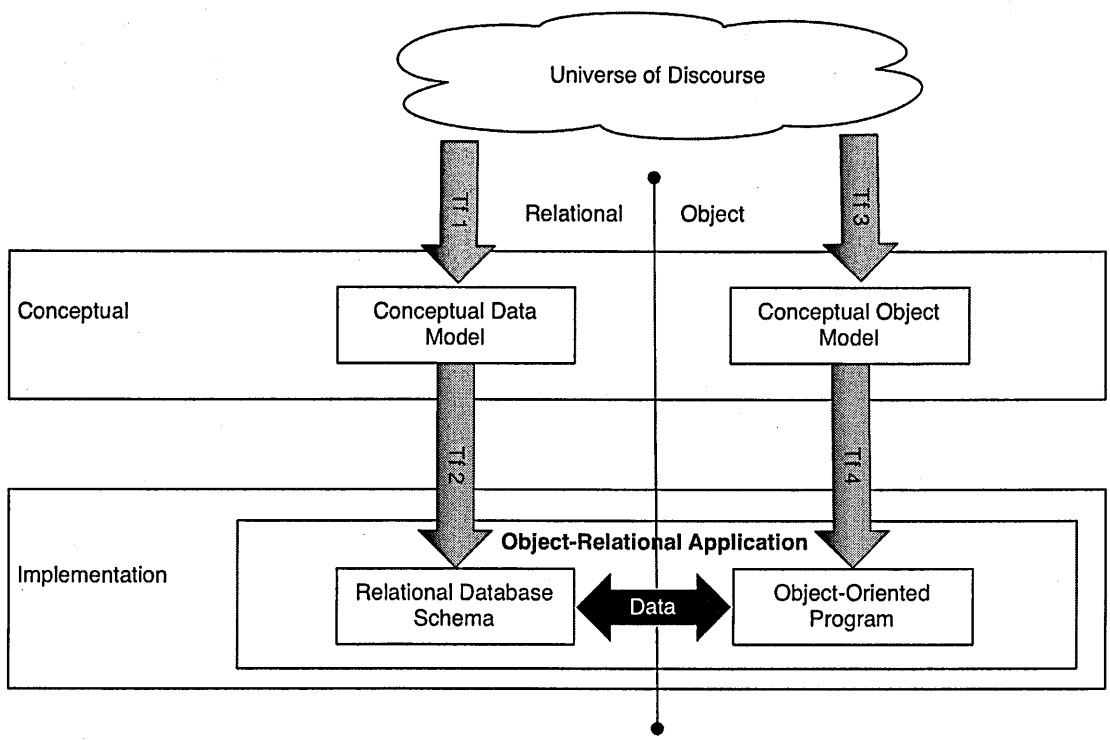


Figure 5 - An Object-Relational Application: The Product of Two Separate Processes

On Figure 5 the context of an object-relational application is shown as a box surrounding a relational database schema and the schema of an object-oriented program. Because, at run-time, an object-oriented program uses a relational database for storage, data must flow between the two and typically in both directions. Such a flow is depicted on Figure 5 as a black arrow labelled "Data".

A mismatch interferes with the seamless transfer of data between an object-oriented program and a relational database. Neward (Neward 2006) characterises a number of mismatches including, what he refers to as, the "object-to-table mapping problem", the "schema ownership conflict", "entity identity issues", the "dual schema problem", the "data retrieval mechanism concern", the "partial object problem" and the "load-time paradox". Two of these mismatches are described here in order to illuminate the problem of object-relational impedance mismatch.

The "object-to-table mapping problem" refers to the apparent correspondence between the data structure of a class in the schema of an object-oriented program, and the data structure of a table in the schema of a relational database. Neward observes that this correspondence quickly breaks down when a mapping strategy is defined between the semantics of an object-oriented class model, such as a hierarchy or an aggregation, and a relational database. For example, a single table or separate tables in the schema of a relational database can represent a class hierarchy or an aggregation.

The "load-time paradox" refers to a problem caused by references between objects. Consider three objects: A, B and C. Object A references object B and object B references object C. So that the reference from object A to object B is valid, in order to create object A it is necessary to first create object B. However, before object B can be created it is necessary to first create object C so that the reference from object B is valid. Consequently when retrieving data for a single object it may be necessary to load data relating to all objects in a network. It may not be practical to load data for all objects in a network into memory so a solution to the problem must be found. Data for an object can be retrieved on demand but this implies that an object can be created with a reference to another object that has yet to be created.

Each characterisation embodies a particular set of concerns that must be addressed if data is to be transferred between an object-oriented program and a relational database. Keller (Keller, Jensen et al. 1993) claims that between twenty and thirty percent of code in an object-relational application is concerned with mismatches. Ambler (Ambler 2003), p106 summarises the consequences of impedance mismatch in practice: "The greater the mismatch between your object and data schemas, the more code you will need to write, test, and maintain". Manifestly the problem of object-relational impedance mismatch can be both costly and time-consuming.

The skill of those responsible for the development of an object-relational application is to find a solution to these mismatches. What is not clear is whether each mismatch is caused by differences between the processes of design, differences between the

languages used, or some other reason. Consequently those responsible for an object-relational application might make an acceptable rather than an appropriate solution.

2.5 Impedance Mismatch Solutions

This section explores the ways in which mismatches have been addressed. There are a number of different solutions. Each solution is described before reflecting on its concern for the cause of a mismatch.

2.5.1 Object-Relational Mapping

If an object-oriented program is to store data in a relational database then a correspondence must be made between artefacts across the two schemas. A correspondence is embodied in a mapping strategy. A transformation is distinct from a mapping strategy.

The concern of a transformation is orthogonal to that of a mapping strategy. A transformation, such as that from an E-R model to a database schema described using SQL, produces a new model. A mapping strategy is concerned with a correspondence between artefacts at the same level of abstraction, that is, between the schema of an object-oriented program and the schema of a relational database.

Generally a concept is described using a collection of artefacts from an implementation language. Typically a subset of artefacts in one language is used to

describe the representation of a concept in another language. A mapping strategy defines a correspondence between the representations of a concept in each language.

In the literature there are many examples of mapping strategies. There are mapping strategies that make a correspondence between a table in the schema of a relational database and a class (for example (Brown and Whitenack 1994)), a hierarchy of classes (for example (Ambler 2003)), an aggregation (for example (Russell 2008)) and an association (for example (Keller 1997) and (Soutou 2001)). Consequently in order to understand that which a database table represents in an object-relational application it is necessary to understand both the correspondence and the schema of an object-oriented program.

Table 1 - Mapping Strategies for a Class Hierarchy

Strategy	Description
One Class One Table	Each class in a class hierarchy corresponds to a separate table in the schema of a relational database.
One Inheritance Tree One Table	All classes in a class hierarchy correspond to a single table in the schema of a relational database.
One Inheritance Path One Table	Each class in a class hierarchy corresponds to a separate table in the schema of a relational database but each table includes columns that represent the attributes of each parent class.

There are different mapping strategies so during the development of an object-relational application a choice must be made. One choice is the mapping of a class hierarchy in the schema of an object-oriented program to a representation in the schema of a relational database. Keller (Keller 1997) describes three mapping

strategies based on SQL:1992: one class one table, one inheritance tree one table, and one inheritance path one table. Each mapping strategy is based on a different correspondence between a hierarchy of classes and the schema of a relational database. Each strategy is summarised in Table 1.

Whilst it is possible to produce a mapping strategy, the correspondence on which it is based may be false or incomplete. Ambler (Ambler 2003), p108 notes that there are subtle differences between a class model and a data model: a data model depicts data structure whilst a class model depicts a structure and within that a definition of both data and processing. Furthermore whilst a class and a table have both intent and extent they are different abstractions. Consequently using a mapping strategy the data of an object is made to fit into a representation based on a row.

Because there is a choice of mapping strategy typically there will be a trade-off (Ambler 2003), p240-243. For example if a table corresponds to a hierarchy of classes then there must be some way to differentiate the class of data in a row. Keller (Keller 1997), p13 suggests that the class of data can be inferred from the value of a column, or a column can be added to a table the values of which would provide a basis for differentiation. The data in an additional column must be maintained and can lead to an increased storage requirement whereas the basis for inferring a class must be clear to those who use a table. The concern of Keller is how to differentiate data in a row rather than why it is necessary to differentiate that data in the first place.

Ambler (Ambler 2003), p223 provides practical guidance in support of a choice of mapping strategy. A choice of mapping strategy has consequences for each schema and the code that must be developed, executed and maintained. Because there are trade-offs a mapping strategy will inevitably involve a compromise. That compromise can be in the schema of an object-oriented program, in the schema of a relational database or in both schemas.

A compromise in the schema of a relational database is demonstrated by the representation of an association between two object classes. An association between two classes can be represented in the schema of a relational database as a relationship between two tables and implemented as a foreign key (Ambler 2003), p250. However the semantics of an association with regard to direction are not the same as those of a foreign key. A foreign key is included in the design of a referenced table; an association between two classes is included in the design of the class that is doing the referencing (Neward 2006).

Shadow information is an example of a compromise in the schema of an object-oriented program. Shadow information is "any data that objects need to maintain, above and beyond their normal domain data, to persist themselves" (Ambler 2006a), p228. Shadow information determines the way an object-oriented program interacts with a relational database. An example of a piece of shadow information is an attribute of an object that indicates whether data has yet to be stored in a database.

Consequently that information is part of the design of an object-oriented program but not part of the design of a database.

In spite of cultural difficulties a mismatch is a shared problem. Meijer (Meijer and Bierman 2011) observes that a mapping strategy can necessitate a change both to the schema of an object-oriented program and the schema of a relational database. For example an object-oriented program must reconstruct data for an object that has been split amongst tables, perhaps as a result of normalisation, and a database will use an index to make such a query execute efficiently.

2.5.2 A Persistence Layer

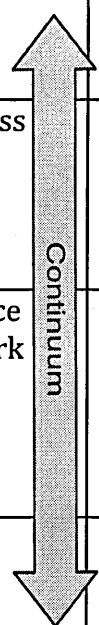
A persistence layer is a part of the design of an object-relational application responsible for the storage and retrieval of data in a relational database. A persistence layer maintains the separate concerns of an object-oriented program and a relational database. It is appealing that somehow a separation of concerns will avoid a mismatch because issues of data storage are detached from those of data processing.

Ambler describes a persistence layer as an implementation of a database encapsulation strategy (Ambler 2003), p199-221. He presents a number of database encapsulation strategies. Each strategy is summarised in Table 2.

A choice of encapsulation strategy typically reflects issues of software development, performance and maintenance. Marguerie (Marguerie 2007) provides a number of criteria for making a choice of products that implement an encapsulation strategy including flexibility, ease of use and support for optimisations. Fussell (Fussell 1997) presents a classification of object-relational mapping sophistication based on the exposure of an object-oriented program to the structure of a relational database schema. He lists different criteria for a choice of solution including performance, cost, scalability, development time and maintenance.

Table 2 - Database Encapsulation Strategies

Encapsulation Strategy	Summary
Brute Force	A program accesses a database directly so there is no encapsulation. Information about the schema of a relational database is spread throughout a program. Technologies include database vendor APIs such as those from Sybase (Sybase), Oracle (Oracle) , Microsoft ODBC (MicrosoftODBC) and JDBC (JDBC).
Data Access Object	Database access logic is contained within a Data Access Object. Information about the schema of a relational database is contained within a Data Access Object. Technologies include Java Data Object (Oracle) and Microsoft ActiveX Data Object (MicrosoftADO).
Persistence Framework	A persistence framework is responsible for access to a relational database. Information about the schema of a relational database is contained within the configuration of a persistence framework. A mapping strategy is used to generate the requisite database access. Technologies include Hibernate (Hibernate) and Oracle TopLink (TopLink).
Service	Ambler (Ambler 2003),p214 defines a service as "an operation offered by a computing entity that can be invoked by other computing entities". A service provides access to data in a database. A program contains no information about the schema of a database. Technologies include Web Services (WebServices) and CORBA (CORBA).



In order to understand the separation of concerns between an object-oriented program and a relational database it is possible to position each of Amblers encapsulation strategies along a continuum. The continuum, shown as an arrow on Table 2, is based on the exposure of the schema of an object-oriented program to information about the schema of a relational database.

At one extreme of the continuum, using a "Brute Force" encapsulation strategy, there is no encapsulation and information about the schema of a relational database is spread throughout the schema of an object-oriented program. At the other extreme of the continuum, using a "Service" encapsulation strategy, the schema of an object-oriented program contains no information about the schema of a relational database. Whilst at this second extreme issues of where and how data is stored are not the concern of an object-oriented program there is a compromise: Ehreke (Ehreke 2005),p1 observes that information about a database has been replaced by information about a persistence layer.

There are a number of benefits from the use of a database encapsulation strategy (Ambler 2003), p200. Moving along the continuum from a "Brute Force" encapsulation strategy to a "Service" encapsulation strategy, a database encapsulation strategy allows a program access to data in a database with less concern for the data structure of that database. Consequently a database encapsulation strategy provides a way to isolate the schema of an object-oriented program from certain changes to the schema of a relational database.

There are also a number of disadvantages from using a database encapsulation strategy (Ambler 2003), p200. Each strategy involves investment in time and effort either to procure and configure or to develop. An encapsulation strategy can "flounder when mappings between object and data schemas become complex" (Ambler 2003), p200, or a strategy can provide too little control over access to a database. Furthermore where there are many applications accessing a database, one application may not use a persistence layer and access the database directly, whilst some functions such as reporting or the bulk loading of data can also bypass a persistence layer.

2.5.3 A Single Conceptual Model

In section 2.4 the model of software development is based on the assumption that a process of object-oriented program design progresses separately from a process of database design. The result is a schema for an object-oriented program and a schema for a relational database that each describes a universe of discourse but in a different way. This section explores the use of a single conceptual object model as the basis for a transformation both to the schema of an object-oriented program and to the schema of a relational database.

It is appealing that somehow the use of a single conceptual object model will avoid a mismatch. Because the design of the schema of an object-oriented program and the design of the schema of a relational database each start from the same conceptual model differences between conceptual models are removed. A conceptual object

model includes a definition of both a data structure and processing and so it can be used in this way. A conceptual data model cannot be used in this way because it is a representation only of data structure.

Figure 6 shows transformations based on the use of a conceptual object model. From a conceptual object model are produced two implementation models. The first transformation (Tf4) takes as input a conceptual object model and produces the schema for an object-oriented program. The second transformation (Tf5) takes as input the same conceptual object model and produces the schema for a relational database.

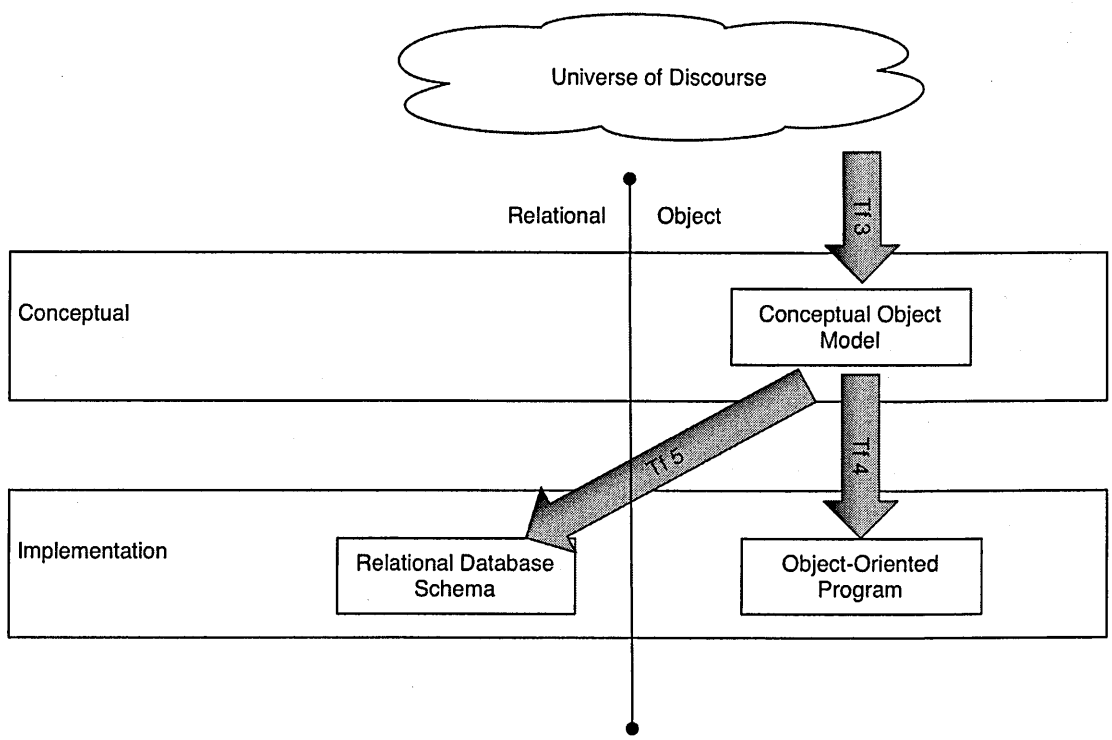


Figure 6 - A Conceptual Object Model used in the Design of Both a Program and Database

Section 2.4.2 concludes that it is possible to produce different schemas for an object-oriented program from the same conceptual object model. However it is also possible to produce different schemas for a relational database from the same conceptual object model. Consequently a correspondence must be made between the artefacts in each schema.

A class model is typically used as the basis for the schema of an object-oriented program, but Marcos (Marcos, Vela et al. 2003), Naiburg (Naiburg and Maksimchuk 2001), and Mok (Mok and Paper 2001) each describe a transformation from a UML class model to the schema of a relational database. However the transformations described by Marcos, Naiburg and Mok each produce a different relational database schema from the same UML class model. For example, Grant (Grant, Chennamaneni et al. 2006) observes that the transformation described by Mok produces a different relational database schema depending on the class in a UML class model at which the transformation starts.

Choices of transformation and differences between implementation languages mean that an object-oriented program and a relational database may each employ a different model of data. Consequently the use of a single conceptual model will result in two different schemas. Because the Model Driven Architecture (MDA) (OMG) employs a single model there are consequences.

The MDA is a four level framework in support of a model transformation. The MDA is concerned with the automation of certain aspects of software development based on a single model. In the MDA a model is a first class citizen (Bezivin 2001) and a single model of a universe of discourse can be transformed into different schemas. Because a single conceptual model does not avoid a mismatch then the MDA will not avoid a mismatch.

2.5.4 A Hybrid Language

Neward (Neward 2007) suggests that one way to eliminate a mismatch is to give relational concepts first-class status in an object-oriented programming language. The data structure of a relational database might somehow be incorporated in an object-oriented program. Meijer (Meijer 2006) and Schwartz (Schwartz and Desmond 2007) each describe the language Microsoft LINQ (MicrosoftLINQ) as one implementation of this idea.

Using Microsoft LINQ to SQL (Microsoft 2011), a Visual Basic (Petroutsos, E. 2010) program can use an SQL-like syntax to query a relational database. Rather than query a database directly in terms of a table, a query is expressed in terms of an entity class. An entity class is part of an object model that is created using the LINQ O/R Designer. This approach, based on an object model, led Meijer (Meijer 2006) to claim that there is no impedance mismatch because a programmer always works in terms of a class not a table.

2.5.5 Reflection

Keller (Keller 1997) observes that impedance mismatch is a fact of life for those who develop an object-relational application. However, in spite of the many solutions, understanding the cause of a mismatch is not a requirement. Typically a solution offers a pragmatic approach to specific problems of implementation.

A mapping strategy is the way a mismatch is addressed but it is not a perfect solution. Typically a mapping strategy involves a compromise so the flow of data between an object-oriented program and a relational database is not seamless. A mapping strategy takes as given a mismatch and provides an answer to the question, how can the data in an object-oriented program be stored in a relational database?

A persistence layer marshals a mapping strategy into a particular part of an object-relational application but does not lead to an understanding of the cause of a mismatch. A persistence framework, such as Hibernate (Hibernate), removes some of the work implementing a mapping strategy, and so addresses some issues of software development cost and time, but does not lead to the question of why a mapping strategy is necessary.

The schema of an object-oriented program and the schema of a relational database are different even when the design of each starts from the same conceptual model. A single conceptual model does not avoid a mismatch and a mapping must be defined

between each schema. Consequently artefacts of a schema are important to understanding a mismatch.

A hybrid language such as Microsoft LINQ does not avoid a mismatch because, whilst a query is expressed in terms of an object model, a mapping strategy is used between the schema of a relational database and an object model. However there might still be a case for sharing other concepts between languages, such as the introduction of object-based concepts in SQL:1999. If the cause of a mismatch is understood then an informed decision about a hybrid language can be made.

2.6 Summary

Object-relational impedance mismatch is a term that is not well defined, covers a range of problems, and for which there are a number of solutions of which none have a requirement for understanding the cause of a mismatch.

In order to understand the cause of a mismatch it is necessary to first understand the problem. The term impedance mismatch was first used in 1984 but there is not a single and cohesive understanding of the problem. Copeland & Maier, Ambler and Neward each use a different language to describe their own interpretation.

An object-relational application provides the context for a mismatch. The development of an object-relational application involves different models, different languages and different people. The result is a schema for an object-oriented program

and a schema for a relational database that each describes the same universe of discourse but in a different way.

If an object-oriented program is to use a relational database for storage then a mismatch between the two schemas must be overcome. A skill of those who develop an object-relational application is to address a mismatch. There are a number of solutions and typically they involve a mapping strategy. A mapping strategy provides an answer to the question, how can the data in an object-oriented program be stored in a relational database. However a mapping strategy typically involves a compromise.

A mapping strategy, a persistence layer, a single conceptual model, the MDA and Microsoft LINQ do not lead to the question of the cause of a mismatch. In order to understand the cause of a mismatch it is important to understand why it is necessary to make a compromise between the schema of an object-oriented program and the schema of a relational database. Because a mapping strategy embodies a compromise understanding a mismatch starts by analysing a mapping strategy.

Chapter 3 Object-Relational Impedance Mismatch

3.1 Introduction

Neward (Neward 2006) refers to the problem of an object-relational impedance mismatch as "a quagmire of issues". This chapter sets out to understand the problem of object-relational impedance mismatch. The objective is to make sense of the problem and the different interpretations of object-relational impedance mismatch.

The problem of object-relational impedance mismatch is characterised as a number of problem themes. Each theme is concerned with a collection of mismatches. In the context of a problem theme it is now possible to talk about the problem of a specific subtype of object-relational impedance mismatch.

Relationships between problem themes are used to visualise the quagmire described by Neward. Exploring relationships between problem themes demonstrates that object-relational impedance mismatch is a complex problem and highlights problems of particular importance.

3.2 A Catalogue of Problem Themes

Chapter 2 concludes that Copeland & Maier (Copeland and Maier 1984), Neward (Neward 2006) and Ambler (Ambler 2003), p105-113 each characterise object-relational impedance mismatch in a different way. Each characterisation draws attention to a collection of mismatches that together represent a particular problem.

In this section the characterisations described by Copeland & Maier, Neward and Ambler along with contributions from others are consolidated as a catalogue of problem themes (Ireland, Bowers et al. 2009a; Ireland, Bowers et al. 2009b).

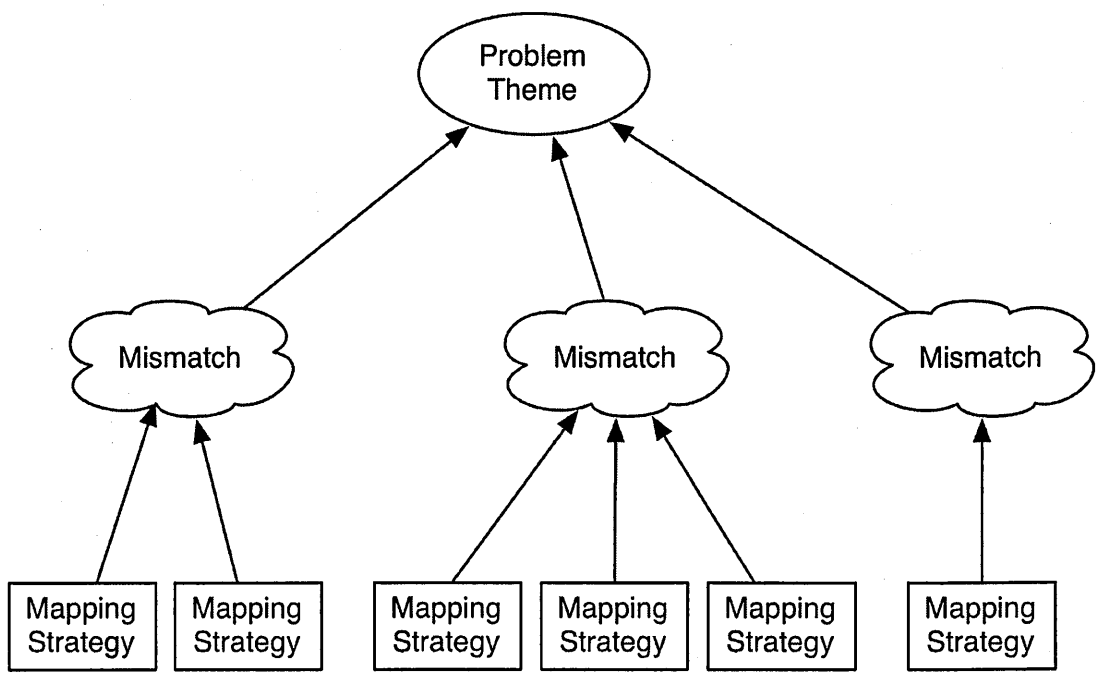


Figure 7 - A Problem Theme, Mismatches and Mapping Strategies

A problem theme is defined as a collection of mismatches. A problem theme reflects a particular characterisation, such as the "object-to-table mapping problem" and the "schema ownership problem" described by Neward or the "cultural impedance mismatch" described by Ambler, and helps to make sense of a collection of mismatches. A mapping strategy is one solution to a mismatch. It follows that a mapping strategy is also one solution to a problem theme. The relationships between a theme, a mismatch and a mapping strategy are summarised in Figure 7.

A problem theme is important for two reasons. A problem theme provides a way to understand one aspect of object-relational impedance mismatch. It is then possible to

talk about and focus on a specific problem rather than use the general term object-relational impedance mismatch. Relationships between themes demonstrate the complex nature of object-relational impedance mismatch and identify themes of particular significance.

The problem themes and their relationships are introduced in Figure 8. Two themes are related if a problem of one theme leads to a problem of another theme. The arrows on each line indicate the direction of influence between two problem themes. It is possible to talk about a specific problem such as that of schema ownership, structure or identity and see that such problems are related. For example the line from the problem theme of structure to the problem theme of identity indicates that a solution to a structure problem can have a consequence for an identity problem.

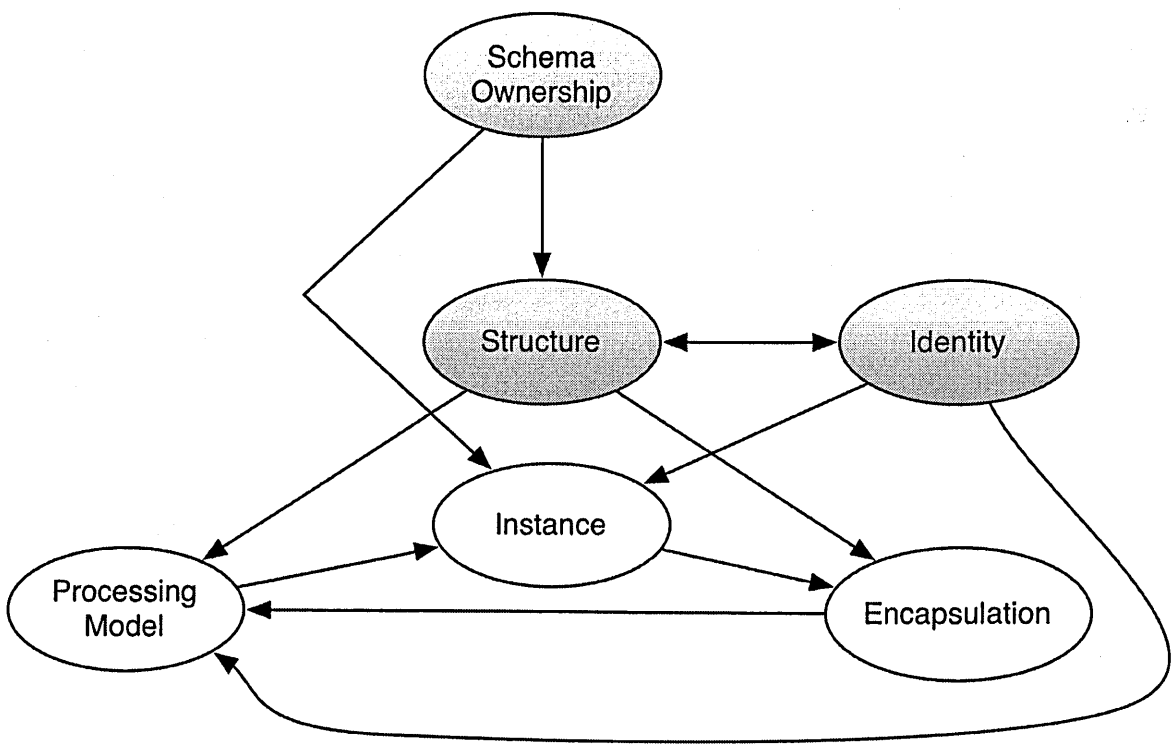


Figure 8 - Problem Themes and Relationships

The following sections describe each problem theme. Each problem theme is illuminated using an example Java program that uses a relational database for storage. The example program is based on the FTI Case Study in Appendix B. The next section introduces the relevant artefacts in the two schemas of this object-relational application.

3.2.1 Example Object-Relational Application

The relational database schema comprises three tables: TRADE, LEG and STOCK. The definition of these tables is shown in Figure 9. The tables TRADE and LEG correspond to the data structure of the class Trade. The table STOCK corresponds to the data structure of class Equity.

The design of Java class Trade is based on the class Trade. Figure 10 shows a fragment of a Java program that creates an object of class Trade from data in the database (Figure 9) and executes that trade on the London Stock Exchange. This example Java program uses the Java Database Connectivity (JDBC) interface. JDBC provides access to a relational database from a Java program and is an example of a Brute Force encapsulation strategy described in section 2.5.2.

```
create table STOCK (
      ISIN              CHARACTER(11) PRIMARY KEY,
      DESCRIPTION       CHARACTER VARYING (100)
      NUMBER_OF_SHARES  INTEGER
)

create table TRADE (
      TRADE_ID    INTEGER PRIMARY KEY,
      STOCK       CHARACTER(11),
      PARTY       CHARACTER(11),
      MARKET      CHARACTER(8)
      CONSTRAINT  stock_fk FOREIGN KEY (STOCK)
            REFERENCES STOCK (ISIN)
)

create table LEG (
      TRADE_ID    INTEGER,
      QTY         INTEGER,
      SIDE        CHARACTER(1),
      PRICE       DECMAL(10,2),
      BOOK        CHARACTER(4)
      CONSTRAINT  trade_fk FOREIGN KEY (TRADE_ID)
            REFERENCES TRADE (TRADE_ID)
)
```

Figure 9 - Relational Database Schema for a Trade

The Java code fragment shown in Figure 10 loads the data for a trade from two database tables: TRADE and LEG, and stores this data in a new object referenced by the attribute `thisTrade`. It then executes the trade on the London Stock Exchange (LSE). Comments in the body of the program provide further details.

```
/* Assume database connection already established */
string sqlCommand = "select t1.STOCK, t1.PARTY,
                     t1.MARKET,
                     l1.SIDE, l1.PRICE, l1.QTY
              from   TRADE t1, LEG l1
              where  t1.TRADE_ID = 1
              and    t1.TRADE_ID = l1.TRADE_ID"
/* Load the details of Trade 1 */
resultSet = jdbcStatement.executeQuery(sqlCommand)
/* Establish an empty in-memory Trade object */
thisTrade = new Trade()
/* Populate the attributes of the Trade object */
thisTrade.setSTOCK(resultSet.getString("STOCK"))
thisTrade.setPARTY(resultSet.getString("PARTY"))
thisTrade.setMARKET(resultSet.getString("MARKET"))
/* Populate the first leg of the trade */
thisTrade.setSIDE(1, resultSet.getString("SIDE"))
thisTrade.setPRICE(1, resultSet.getFloat("PRICE"))
thisTrade.setQTY(1, resultSet.getFloat("QTY"))
thisTrade.setBOOK(1, resultSet.getString("BOOK"))
if (resultSet.next()) {
      /* Two legged trade - not all are like this */
      thisTrade.setSIDE(2, resultSet.getString("SIDE"))
      thisTrade.setPRICE(2, resultSet.getFloat("PRICE"))
      thisTrade.setQTY(2, resultSet.getFloat("QTY"))
      thisTrade.setBOOK(2, resultSet.getString("BOOK"))
}
/* Object thisTrade is now populated */
/* Execute the trade on the London Stock Exchange */
thisTrade.executeOn(LSE)
/* Consideration will have been changed by charges */
thisTrade.calculateConsideration()
```

Figure 10 - Executing a Trade

3.2.2 Structure Problem Theme

Copeland & Maier first recognised a difference of structure as one source of an impedance mismatch. For them a structural mismatch is a concern for a difference of data type. Neward refers to a difference of structure using very specific language as "the object-to-table mapping problem". However under this heading he then describes a number of other problems of this kind, such as the representation of both a hierarchy of classes and an association between classes.

The structure problem theme is concerned with any difference of data structure between the schema of an object-oriented program and the schema of a relational database, and so adopts a broad interpretation of issues of structure. The essence of a structure problem is the extent to which an object-oriented data structure can be, and should be described by a relational data structure. Problems of the structure theme are important because they are concerned with a description of the data processed by an object-relational application.

The building blocks of an object-oriented language such as Java are different to those of SQL:1992. Using Java a programmer defines a structure in terms of a class. Using SQL:1992 a structure is defined in terms of a table. However a class and a table are different abstractions.

A class has both an arbitrary structure in terms of attributes and an arbitrary semantics defined in potentially many methods. A class may be part of a hierarchy of classes and so will include and possibly modify the semantics of a parent class. A table has a structure based on a column. Each column has a domain limited to one of a predefined data type. A column of a row can hold a single value. Such a difference of abstraction means that data about an object will not necessarily fit neatly into a row of a table.

In the FTI Case Study a trade can have more than one leg. A house equity trade will have one leg that will be either a Buy or Sell. A book transfer trade will have two legs,

one representing a Buy from Book "A" and the other a Sell to Book "B". Each leg of a trade has a different value for columns SIDE, PRICE, BOOK and QTY. The values of columns SIDE, PRICE, BOOK and QTY are stored in a row of the database table LEG. So whilst a trade and the legs of a trade are described as a single class Trade in the schema of an object-oriented program, in a database two tables are used to represent a trade.

When executed, the SQL query defined in `sqlCommand` in Figure 10 will return one row for a house equity trade but two rows for a book transfer trade. This structure problem has been addressed at the schema level using a mapping strategy that involves processing the next row of a result if one exists. The mapping strategy is implemented by the statement `if(resultSet.next()).`

3.2.3 Instance Problem Theme

The essence of an instance problem theme is where the canonical copy of state is located. Problems of the instance theme are important because they are concerned with the ownership and the responsibility for data.

Rosenberger in (Marinescu and Zicari 2008) make the distinction between a client and a server. For Rosenberger an object-oriented program may be thought of as a client and a relational database as a server. He refers to a problem of this theme as a problem of state. He observes that a relational database works "as if there would be no state on clients". In other words a database holds the canonical copy of state.

However an object-oriented program also has a state, and that state may change and so no longer be the same as the state recorded in the database.

Differences in perception of the role of a program and a database bring into question the location of a canonical copy of state. A database designer can understand that a database holds the canonical copy of state because a database has typically been used in this way and data is shared amongst applications.

Those responsible for an object-oriented program can understand that a program holds the canonical copy of state if, for example, certain data is not stored in a database but is derived. Also a database can be used to store only some of the data and then only at the end of a day, or a program can use a cache of data (Neward 2006) and so not guarantee to store each change as it happens.

In the FTI Case Study the value of the consideration of a trade can be derived from the product of price and quantity, respectively columns PRICE and QTY in the table LEG. As a consequence the consideration of a trade is not stored explicitly in the table LEG. There is no column in the table LEG to store the value of a consideration and so a programmer might believe that an object holds the canonical copy of state. This instance problem is represented in the Java program, at the schema level, by the method `calculateConsideration` of the class Trade.

3.2.4 Encapsulation Problem Theme

The principle of encapsulation requires that the state of an object can only be determined by its behaviour, so in an object-oriented program the value of an attribute of an object is accessed via a method. Problems of encapsulation are important because in a database the value of a column in a row has no such protection. Consequently, once stored in a database, data may be changed without the protection of those semantics encoded in a method.

Copeland & Maier are not concerned with issues of encapsulation because they base their characterisation on a procedural language that does not support this concept. Ambler (Ambler 2003), p96 considers encapsulation as a mechanism for decoupling an object-oriented program and a relational database. Lodhi (Lodhi and Ghazali 2007) demonstrates an encapsulation problem in the context of an object. They observe that the way an association is represented in a relational database, using a foreign key, is opposite to a relationship between the two objects. Consequently the principle of encapsulation is not preserved when an association between the two objects is stored in a database.

In the example Java program (Figure 10) the method `executeOn` causes data about a trade to be sent to a stock exchange. Such an interaction with the London Stock Exchange modifies the values of some of the attributes of a trade object. For example, after execution (i.e. `thisTrade.executeOn(LSE)`) a trade object will have an exchange trade identifier and some execution charges. Once stored in the database these

charges may be modified independently of the Java program and so may no longer reflect the formula by which they were derived in the method `executeOn`.

3.2.5 Identity Problem Theme

The essence of an identity problem is how to identify uniquely a collection of data values between both object-oriented program and a relational database. Such problems of identity are important to ensure the integrity of data between an object-oriented program and a relational database.

The semantics of identity are different between an object-oriented program and a relational database. An object has an identity independent of its value so two objects with the same value are different if they have a different identity. Across two executions of a program data will be held in a different object. A primary key is an identity for rows of a table. Each row of a table must have a unique value for a primary key. Typically the value of a primary key does not change whereas the identity of an object will be different between two executions of a program.

Neward (Neward 2006) refers to the identity problem as "entity identity issues". He considers the symptoms of an identity problem as they materialise as issues of integrity. Because an object for processing is separate from data in a database and each has its own identity, in order to accurately reflect a change to the value of an object in a database first the correct data must be identified in that database. Issues of integrity between an object-oriented program and a relational database are

concerned with the canonical copy of state. Consequently an identity problem can have a consequence for an instance problem.

Ambler (Ambler 2003), p285 includes issues of identity when considering the performance of an object-relational application. He refers to the Identity Map pattern of Fowler (Fowler, Rice et al. 2003). This pattern describes a cache of objects used to prevent the duplicate retrieval of data about the same object from a database. A duplicate object is a choice of design and can lead to an instance problem because each duplicate must reflect changes consistently.

In the example Java program (Figure 10) the statement `thisTrade = new Trade()` creates a new in-memory trade object and assigns `thisTrade` a pointer to it. A Java runtime environment will assign the identity of this new object.

The identity of an object of class Trade and a value of column TRADE_ID cannot be assumed to be the same. The identity of a trade in the database is the primary key TRADE_ID. The value of the column TRADE_ID of the trade being retrieved from the database using `sqlCommand` is "1". The value of the identity of an object will be a pointer to a location in memory or some other reference.

The column TRADE_ID is purely a database concern however, in order to resolve this identity problem, the Java program must retain the value in column TRADE_ID retrieved from the database. There is a choice of how such a correspondence is

maintained. In Figure 10 identity has been hard-coded for brevity but an attribute of an object could have been used or an object cache such as that described by Fowler.

3.2.6 Processing Model Problem Theme

The essence of a processing model problem is how to represent, maintain and retrieve from a database a sufficient set of objects for processing. Such problems are important because they concern issues of software performance (Keller 1997).

Problems of this theme occur because of a difference in the model of data access between an object-oriented program and a relational database. An executing object-oriented program may be thought of as a network of interacting discrete objects. The model of access can be thought of as navigation. The relational model is declarative and the model of access is based on a set.

Neward (Neward 2006) refers to "the data retrieval mechanism concern", "the partial object problem" and "the load time paradox" and all three are part of the processing model problem. His concern for data retrieval is how to query a database for data relating to an object. He describes a number of options for how to query a database but in essence the issue is one of identity. Thus a solution to an identity problem can cause a processing model problem.

The partial object problem arises because of a concern for performance over a network. Ideally, only that data required to complete some processing is retrieved

from a database. However in order to create an object sufficient data must first be retrieved from a database. Consequently there is a trade-off because, in order to create an object, it may be necessary to retrieve more data than is necessary to complete the processing.

The load-time paradox is concerned with a network of references between objects. Because an object can reference other objects, before an object can be created it may be necessary to first create those referenced objects. In the example Java program (Figure 10) the data relating to a particular equity are stored in a row of the database table STOCK. There is a foreign key in table TRADE, referencing table STOCK. A trade must be for an instrument represented as a row in table STOCK. Consequently when retrieving data for an object of class Trade a decision must be made regarding the reference to an object of class Equity.

The Java method `setSTOCK` (e.g. `thisTrade.setSTOCK(resultSet.getString ("STOCK"))`) can take the string value of the STOCK column, returned by JDBC, for a row of table TRADE and convert this into a Java pointer to an object of class Equity. Alternatively the function can just hold the string value of the STOCK column if it is not necessary to refer to an equity object.

In order to resolve this processing model problem first problems of structure and identity must be resolved. In order to create an object of class Equity data must be retrieved from the table STOCK. In order to avoid a duplicate in-memory object the

Java program must handle the scenario whereby data for a row of table STOCK object has already been loaded into an object. This scenario can occur if previously a trade has been made in a particular equity.

3.2.7 Schema Ownership Problem Theme

The essence of the schema ownership problem is that the team who design and implement an object-oriented program can be a different team from the team who design and implement a relational database. Such problems are important because they concern the choices made by those responsible for an object-oriented program and a relational database.

Chapter 2 established that each team has a different agenda and will make different choices such as those of a language and an abstraction. Ambler (Ambler 2003), p111 uses the term a "cultural impedance mismatch" to refer to differences of agenda between the teams responsible for an object-oriented program and a relational database. The consequences of a cultural impedance mismatch for Ambler are project failure, increased staff turnover, and that other problems such as those of structure are exacerbated because the two teams have difficultly working together.

Neward (Neward 2006) describes issues of "schema-ownership conflict" and " the dual schema problem". Each reflects a different aspect of the schema ownership problem. The dual schema problem is concerned with the consequences of a change to a schema such as a change to the definition of a table. The essence of a schema

ownership conflict is that "the database schema itself is not under the control of [program] developers".

A solution to a schema ownership problem must involve both teams. Neward argues that a development team start a new application so they should be responsible at least for an initial database design. Those responsible for a database only need to be involved when there are issues of performance or when data is used by another application. However Russell in (Marinescu and Zicari 2008) notes that issues of performance can occur when a relational database schema is generated from the data structure of an object-oriented program. Consequently it is not safe to assume that those responsible for an object-oriented program understand the process of database design. Also those responsible for the design of a database might not appreciate the consequences, for an object-oriented program, of a particular change to a database schema.

A programming team is responsible for the Java code in Figure 10. A database team is responsible for the database schema in Figure 9. To produce the SQL statement defined in `sqlCommand` requires knowledge, on the part of a programmer, of the relational database schema. Furthermore the execution of `sqlCommand` returns a result in which the columns SIDE, PRICE, BOOK and QTY are repeated for each leg of a trade. These semantics are not apparent from the SQL statement in `sqlCommand` and a programming team must understand to which LEG of a trade a row corresponds.

In order to resolve this schema ownership problem a programming team needs to either understand the database schema, the SQL and the result or request help from the database team. In the latter case, both a programming team and a database team need to be able to understand the correspondence between their respective models.

3.3 A Complex Mix of Problems

Problem themes are important because they concern mismatches that must be addressed during the development of an object-relational application. However such concerns are not independent. This section explores relationships between problem themes. Some of the relationships have already been exposed but there are others. Each relationship is causal; it is important to understand these relationships because collectively they describe the complex nature of object-relational impedance mismatch.

A structure problem can be the result of an ownership problem. Chapter 2 observes that issues of a conceptual mismatch and a structural mismatch as described by Copeland & Maier are not independent. A conceptual framework determines the semantics of a language and so a language such as Java is referred to as an object-oriented language. Those who implement an object-relational application make a choice of abstraction and use the artefacts of a particular language to describe that abstraction.

Neward (Neward 2006) refers to "the schema ownership conflict" and "the dual schema problem". Each demonstrates a relationship between a schema ownership problem and a structure problem. The schema ownership conflict describes a mismatch of agenda. For example an issue of program performance means that those responsible for a database may have to change a data structure (Sanders and Seungkyoon 2001). The dual schema problem occurs when a database must be changed in order to accommodate another application. Consequently a structure problem is caused by a solution to a schema ownership problem.

A choice of abstraction made in the design of another application can produce a structure problem. Keller (Keller 1997) helps to make a link between the schema ownership problem and the structure problem. Whilst Neward is concerned with accommodating a new application, for Keller the problem is incorporating an existing data structure, from another application, into the schema of an object-oriented program.

A schema ownership problem can cause an instance problem. Differences in perception of the role of a program and a database, such as those introduced in section 3.2.3, bring into question the location of a canonical copy of state. A solution to a schema ownership problem must reconcile these different perceptions.

Problems of structure and identity are related. A choice of language will decide the data structure to which an identity refers. For example, in Java an object has an

identity whilst in SQL a primary key provides the identity of a row. So that a solution can be found, in order to address an identity problem it is important to be clear about the structure to which an identity refers.

A solution to an identity problem can cause a structure problem. Keller (Keller 1997) describes a solution to a correspondence of identity between the schema of an object-oriented program and the schema of a relational database. He addresses an identity problem by introducing a surrogate identifier resulting in the need for a change to the structure of each schema.

An instance problem can cause an encapsulation problem. Section 3.2.4 observes that once the charges applied to a trade are stored in a database those charges may be modified independently of the formula used in the method `executeOn`. Such a change can occur if those responsible believe that a database maintains the canonical copy of state.

A structure problem can lead to an encapsulation problem. Lodhi (Lodhi and Ghazali 2007) observes that the way an association is represented in a relational database can be contrary to an association between two objects. Consequently the representation of an association between objects as a foreign key in a relational database does not necessarily preserve the encapsulation of an object.

A structure problem can also lead to a processing model problem. The process of normalisation can cause data about an entity to be split across a number of tables. Consequently, in the context of a reference between two objects, in order to retrieve the data for a referenced object it may be necessary to join a number of tables.

An instance problem can be caused by a processing model problem. It may not be necessary to retrieve or store all the data about an object in order to satisfy a request. However it may still be necessary to retrieve all the data for an object in order to create that object. Consequently those responsible for an object-oriented program might believe that a program maintains the canonical copy of state.

An encapsulation problem can lead to a processing model problem. In order to reference an object, a program must first create an object. It may not be desirable or practical to load data about all objects in a network from a relational database so a decision must be made at which point to stop. That decision is difficult because the network of references between objects is encapsulated within the objects themselves.

3.4 Reflection

The problem of an object-relational impedance mismatch can be characterised using a number of problem themes. These problem themes represent a consolidation of the work of Copeland & Maier, Ambler and Neward. Each theme represents a collection of mismatches and the importance of each theme has been established.

Using the language of problem themes it is possible to be specific about the problem of object-relational impedance mismatch. It is now possible talk about a structure problem or an identity problem rather than the overarching term an object-relational impedance mismatch.

3.4.1 Relationships Between Problem Themes

Problem themes are related and these relationships are summarised in Figure 8. Because it is possible to make a connection between themes it is also possible to explore the complex nature of object-relational impedance mismatch. For example in order to address an identity problem it might be necessary to first address a structure problem, but a structure problem might be caused by a schema ownership problem. By exposing such relationships between themes it is now possible to begin to understand the problem of object-relational impedance mismatch in a meaningful way.

Because there are relationships between themes a solution to one problem will fix that problem but it may also cause another problem. The implication is that a problem must not be considered in isolation. In order to understand a problem of object-relational impedance mismatch first the impact of relationships between themes must be understood.

Figure 8 shows that a structure problem can be caused by a schema ownership problem. Consequently it is important that a solution to a structure problem involves

both those responsible for an object-oriented program and those responsible for a relational database.

Figure 8 also shows that a schema ownership problem is related to a structure problem but a structure problem has a consequence for a number of other problem themes. Similarly an identity problem has a consequence for a number of problem themes. The identity problem and the structure problem are also related. An understanding of the structure problem, the schema ownership problem and the identity problem is therefore important to understanding object-relational impedance mismatch.

There is a cycle in Figure 8. A solution to a structure problem can cause an encapsulation problem. A solution to an encapsulation problem can cause a processing model problem. A solution to a processing model problem can cause an instance problem. A solution to an instance problem can cause another encapsulation problem. Because there are different solutions to a problem a choice of solution must be made.

A choice of a mapping strategy can break a cycle because that solution does not cause another problem. For example Shadow Information (Ambler 2003), p228 introduces a change in the structure of an object-oriented program that addresses an instance problem. The implication is that it is important to understand the consequences of a mapping strategy as well as the artefacts involved in a correspondence.

The importance of a structure problem is reflected in the many mapping strategies between the schema of an object-oriented program and the schema of a relational database. Many authors describe a correspondence of structure between the schema of object-oriented program and the schema of a relational database. A mapping strategy is based on a perceived correspondence such as that between a class and a table (for example (Brown and Whitenack 1994), (Philippi 2005) and (Lodhi and Ghazali 2007)); a class hierarchy and a table or a collection of tables (for example (Ambler 2003), (Hohenstein 1996), (Cabibbo and Carosi 2005) and (Pizzo 2007)); a relationship and a foreign key or a table (for example (Ambler 2003), (Brown and Whitenack 1994), and (Cabibbo and Porcelli 2003)); and an aggregation and a table or a column (for example (Russell 2008) and (Keller 1997)). In order to understand such a choice of mapping strategy and whether it results in an appropriate or an acceptable solution first the cause of a mismatch of structure must be understood.

In order to address a mismatch it may be necessary to address another problem first. A Synthetic Object Identity (Keller 1997),p21 is a surrogate identifier used in a number of mapping strategies. In order to address an identity problem a Synthetic Object Identity introduces a change of structure. The question remains whether this change of structure addresses the real cause of a mismatch of identity, and so is an appropriate solution, or whether the change of structure deals with the symptoms and so is an acceptable solution. To answer that question first the cause of a mismatch of identity must be understood.

3.4.2 The Limitations of Problem Themes

The problem themes represent a consolidation of the work of others. However it is not clear if they identified all possible problems and explored all possible relationships, or whether that was in fact their objective. It is also not clear from the literature whether their categorisations of the problem are simply observations based on experience or an exhaustive search of the problem.

Copeland & Maier talk in general terms of concept and structure, whereas Neward is concerned with specific problems such as retrieving data for an object from a database. Whilst the categorisation of Neward appears more comprehensive than that of Copeland & Maier because it describes more problems, the level of abstraction can explain such a difference. Consequently the catalogue of problem themes, the relationships between themes, and the categorisations of Copeland & Maier, Neward and Ambler must be considered as partial but demonstrative of the problem of object-relational impedance mismatch.

Relationships between problem themes cannot be used to locate the cause of a mismatch. In order to locate the cause of a mismatch it is necessary to first explore the reason for that mismatch between the schema of an object-oriented program and the schema of a relational database. The reason for a mismatch does not lie in a relationship between two problem themes. For example, the answer to an identity problem is not found by understanding that it is caused by a mismatch of structure. First why there is a mismatch of structure must be understood.

Chapter 2 described some of the choices of transformation that provide the context for a mismatch. One mismatch is that of a data structure. Differences of language and abstraction lead to such a mismatch but a conceptual framework underpins each language and each abstraction. Using problem themes it is clear that a problem of structure can have consequences for other problem themes but it not clear whether a choice of abstraction, language or conceptual framework is the root cause of a mismatch of structure.

3.5 Summary

Copeland & Maier, Neward and Ambler each describe different aspects of the problem of impedance mismatch using different terms and at different levels of abstraction. The catalogue of problem themes consolidates their work and gives a structure to the range of problems they describe. It is now possible to refer to a specific problem such as structure, identity or schema ownership rather than the general term an object-relational impedance mismatch.

A problem theme is the label used for a collection of mismatches that characterise object-relational impedance mismatch. Relationships between problem themes demonstrate that object-relational impedance mismatch is a complex problem. Problems of structure and identity are important to an object-relational impedance mismatch but such problems are ultimately set in the context of problems of schema ownership.

Whilst a mapping strategy will address a mismatch, from the relationships between problem themes it emerges that a choice of a mapping strategy is not made in isolation but has a consequence for other problems. For a given mismatch there is also a choice of mapping strategy and by implication a choice of how to address a problem.

Each problem theme focuses attention on a different aspect of a mismatch. However relationships between themes cannot be used to identify the cause of a mismatch. Of particular importance are mismatches of schema ownership, structure and identity because an understanding of these will provide a foundation for understanding other problems. An understanding of the cause of a mismatch provides the motivation for the framework and classification presented and explored in the following chapters.

Chapter 4 The Framework and Classification

4.1 Introduction

This chapter introduces a framework ("the framework") as a way to identify the cause of a mismatch. The framework (Ireland, Bowers et al. 2009a) comprises four levels of abstraction based on a consolidation of the concerns of Copeland & Maier, Neward and Ambler. Each abstraction level cuts across both object and relational technologies.

At each level of abstraction a particular kind of object-relational impedance mismatch is identified. This classification is distinct from the catalogue of problem themes. Associated with each kind of impedance mismatch is a particular kind of dialogue. A dialogue is concerned with understanding and reconciling a difference between artefacts at a particular level of the framework.

The consequences of the framework for the definition of a mapping strategy are explored. There is an opportunity to improve the way that a mapping strategy is described. The framework is compared and contrasted with that of Fussell (Fussell 1997). The framework and that of Fussell are concerned with a different objective. Fussell is concerned not with the cause of a mismatch but where in an object-relational application a mismatch should be addressed.

4.2 The Framework

Chapter 1 observed that the concerns of Copeland & Maier, Ambler and Neward could be understood as four levels of abstraction. The data of an executing application, a schema, a language and a conceptual framework are different levels of abstraction. A conceptual framework provides a context for understanding a language that in turn provides a context for understanding a schema. A schema provides a way to understand the data of an executing application. Table 3 summarises the correspondence between the concern of each author and each level of abstraction.

Table 3 - Four Levels of Abstraction

Level of Abstraction	Author(s)
Conceptual Framework	Copeland & Maier – Concern for issues of paradigm.
Language	Copeland & Maier – Concern for issues of data type.
Schema	Ambler and Neward – Concern for the design of an object-relational application.
Executing Application	Neward – Concern for specific data access and storage problems.

This chapter presents a framework that is based on the four levels of abstraction: concept; language; schema and instance. The hypothesis tested in the following chapters is that a framework based on these four levels of abstraction can be used to identify the cause of a mismatch. The framework is summarised in Figure 11.

At each level of the framework there is a concern for a particular collection of artefacts. A difference between these artefacts can be the cause or a symptom of a

mismatch. If a difference is a symptom of a mismatch then that mismatch is appropriately addressed at another level of the framework.

Object and relational silos span all four levels of the framework. Within each level there are therefore both object and relational artefacts. Each *silo* represents a separation of concerns between technologies based on the concept of an object and technologies based on the concept of a relation. Artefacts based on the concept of an object are shown as a silo down the left-hand side of Figure 11 whilst artefacts based on the concept of a relation are shown as a silo down the right-hand side of Figure 11.

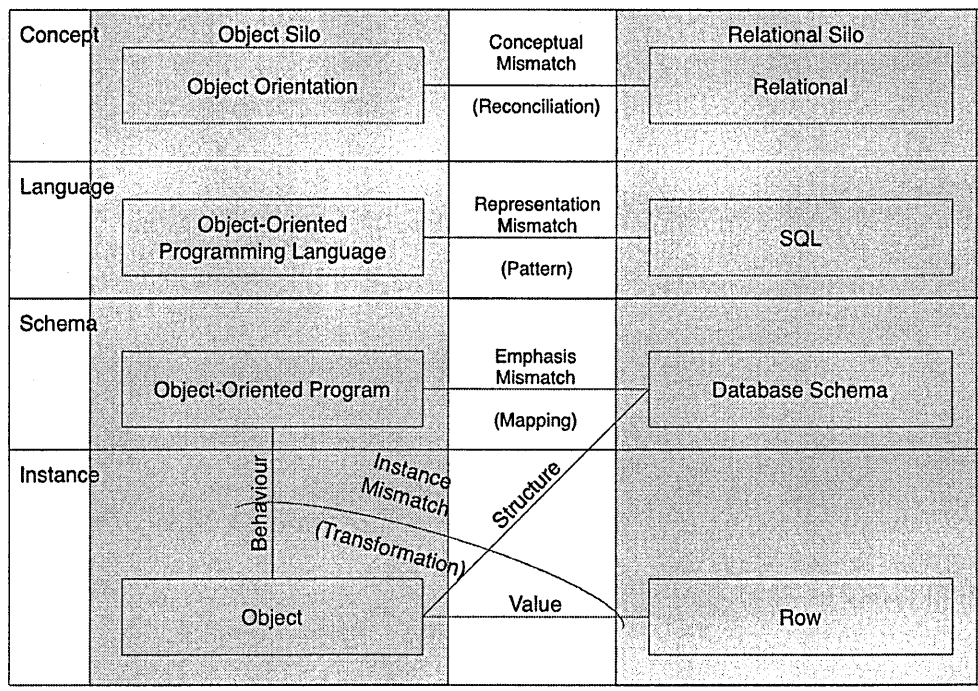


Figure 11 - The Framework

A silo is comprised of artefacts in each of the four levels of the framework. For example, consider the relational silo. At the instance level the relational silo includes a row in a table. At the schema level the relational silo includes the definition of a table

in the schema of a relational database in which that row is stored. At the language level the reference silo includes the definition of the concept of a table in the SQL language on which the definition of that table is based. Finally at the concept level the relational silo includes the concept of a relation on which the semantics of a table in the SQL language are based.

The relationship between the levels of the framework is one of context. A conceptual framework sets the context for the semantics and the artefacts of a language. A language provides both a data and a processing structure for describing the semantics of a universe of discourse in the form of a schema. A schema sets the structure into which data about some thing from a universe of discourse must fit.

The levels of the framework are labelled using terms that may themselves have an alternative interpretation and therefore require clarification. A conceptual framework, represented by the concept level, is one particular way of viewing a world. A language is used to produce a description of a universe of discourse as a schema. A schema is a description of a universe of discourse, expressed using a particular implementation language. In this dissertation program source code is considered the schema for an object-oriented program just as an SQL script is the schema for a relational database. Finally an instance is data about some thing from the universe of discourse set within a particular schema as part of an executing application.

The context of a level of the framework has implications for choices made during the development of an object-relational application. In terms of the framework an implementation language brings with it not only an implicit choice of a conceptual framework but also a particular way to describe data and processing. The objective is to establish, using the framework, whether a mismatch between two schemas is a consequence of a choice of language, a choice of design that is a choice of how the language is used, or a consequence of the implicit choice of conceptual framework when the language was chosen.

At each level of the framework emerges a particular kind of object-relational impedance mismatch: at the concept level a conceptual mismatch; at the language level a representation mismatch; at the schema level an emphasis mismatch; and at the instance level an instance mismatch. Each kind of object-relational impedance mismatch gives rise to a specific concern for a difference between artefacts.

Each concern is addressed through a dialogue, shown in brackets on Figure 11: at the concept level a conceptual mismatch is addressed through a reconciliation dialogue; at the language level a representation mismatch is addressed through a pattern dialogue; at the schema level an emphasis mismatch is addressed through a mapping dialogue; and at the instance level an instance mismatch is addressed through a transformation dialogue. Each kind of mismatch and each kind of dialogue is concerned only with artefacts at a single level of abstraction. However because of the

relationship of context between the levels of the framework, the product of a dialogue can have consequences at another level.

A dialogue involves an exchange of ideas, across silos, about a mismatch. A dialogue is concerned with the reason for a mismatch; in other words understanding why it is that data in an object-oriented program will not fit neatly into a relational database. Because it is an exchange across silos, a single person can undertake a dialogue as long as they understand the issues in both silos. The product of a dialogue might be a new basis for a correspondence, a change to an existing mapping strategy or a change to the design of a schema.

The catalogue of problem themes is distinct from the classification of object-relational impedance mismatch. A problem theme is the label used for a collection of mismatches between the schema of an object-oriented program and the schema of a relational database. Each kind of object-relational impedance mismatch provides the motivation for a dialogue across silos at a particular level of the framework. Figure 12 demonstrates the relationship between a problem theme, a mismatch, a mapping strategy and the framework.

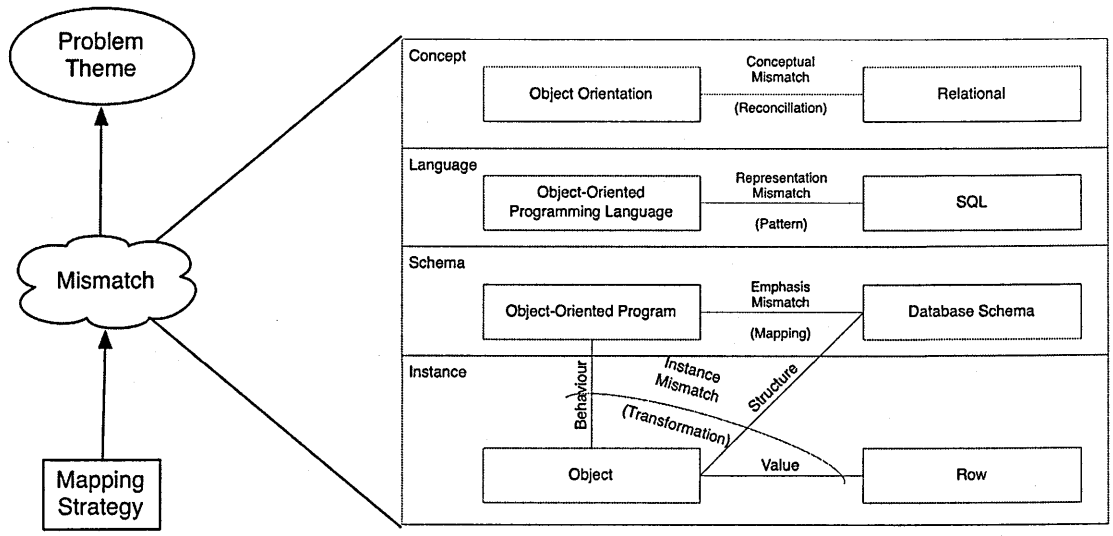


Figure 12 - Exploring the Cause of a Mismatch

For each mismatch there can be a choice of mapping strategy. Figure 12 shows one mismatch and a mapping strategy that addresses that mismatch. The concern of the framework is to understand whether a mapping strategy is an appropriate solution to a mismatch. In other words whether a mapping strategy addresses a mismatch at an appropriate level of abstraction.

All the levels of the framework are used to identify the cause of a mismatch. The cause of a mismatch can be at any level of the framework. Once the cause has been identified it is then possible to understand whether a mapping strategy is an appropriate solution. Figure 13 shows the cause of a mismatch which has been identified at the language level and which relates to a correspondence between artefacts in an object-oriented programming language and SQL. An appropriate solution would address this representation mismatch. An acceptable solution would

address the symptoms of this mismatch at another level of the framework. Figure 13 demonstrates an acceptable solution that involves a change to the design of a database schema at the schema level.

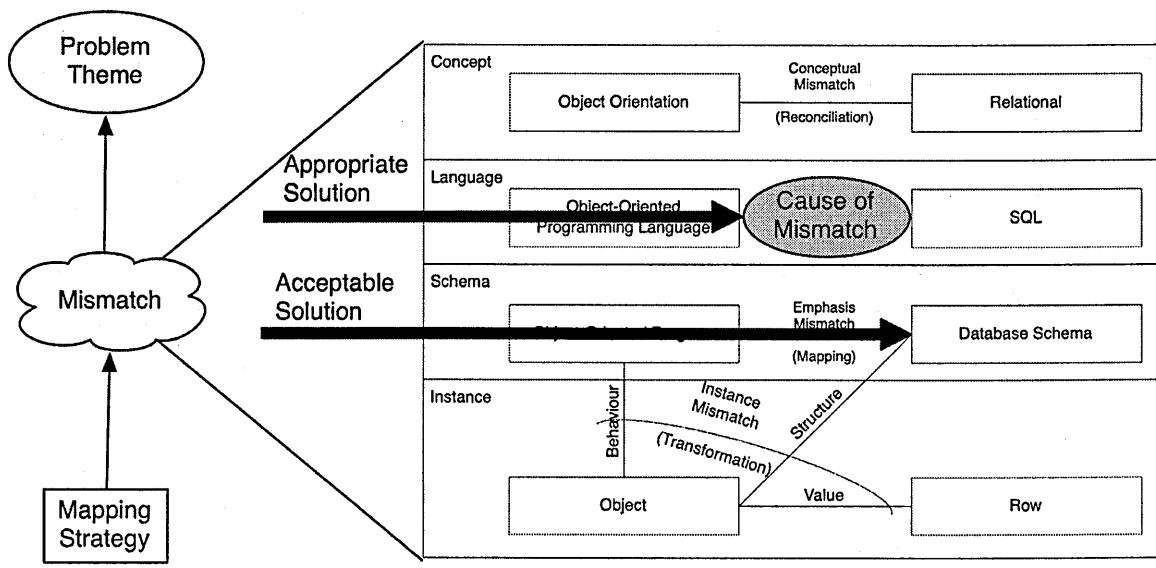


Figure 13 - An Acceptable and an Appropriate Solution

Understanding a mapping strategy involves engaging in a dialogue at each level of the framework. The objective of a dialogue is to address why a mismatch occurs, the underlying assumptions of a mapping strategy and any compromises and their consequences that a mapping strategy involves.

A mapping strategy is distinct from a dialogue. A dialogue involves an exchange of ideas, across silos, about a mismatch. A mapping strategy, such as those described by Ambler (Ambler 2006a), Brown (Brown and Whitenack 1994) and Keller (Keller 1997) is concerned with a correspondence between the schema of an object-oriented

program and the schema of a relational database; in other words how best to make data in an object-oriented program fit into a relational database.

4.3 The Levels of The Framework

The framework provides a new way to think about a mismatch and how to go about addressing it. This section describes each level of the framework and explores the artefacts and the concerns of the corresponding dialogue.

4.3.1 Concept Level

A dialogue at the concept level is concerned with reconciling different ways of thinking about a world. In discussing an object-relational application, one perspective is based on object concepts and in this dissertation is referred to as the object framework. The other perspective is based on relational concepts and in this dissertation is referred to as the relational framework.

The object framework and the relational framework use a different set of concepts to describe a world. Chapter 2 established that there is no single agreed definition of an object, but the object framework will typically include concepts such as class, subclass, object, attribute, and association. There is a single definition of what constitutes the relational framework. The relational framework includes concepts such as relation, tuple and domain. The relational model (Codd 1970) is a way to conceive of the relational framework and the semantics are those of set theory.

It is possible to make a correspondence between object and relational concepts. A class is a type definition that is similar to the intention of a relation. An instance of a class, being an aggregation of values, is similar to a tuple of a relation. However there is not a direct correspondence between the extent of a relation, as a collection of tuples, and all instances of a class.

The extent of a relation is a set but the relationship between an instance and a class is not one of set membership. In order to correctly describe the semantics of a relation in the object world it is necessary to introduce a new concept of a collection class such as a set or a bag. A parallel can then be drawn between the semantics of an instance of that class and a relation because that instance can contain other objects.

A lack of correspondence between two perspectives on a world materialises as an object-relational impedance mismatch. This kind of object-relational impedance mismatch is labelled a conceptual mismatch. A reconciliation dialogue is concerned with a conceptual mismatch.

A reconciliation dialogue will explore differences in perspective, terminology and semantics. The introduction of a collection class is one example of reconciliation and Date in (Kalman 1994) provides another. He uses the example of a class and a domain to emphasise that relational theory is not at odds with the ideas of object-orientation. He argues that just as the semantics of a class are arbitrary, the relational model does

not prescribe a domain. For Date the correct correspondence is between a class and a domain and not between a class and a relation.

4.3.2 Language Level

A dialogue at the language level is concerned with understanding differences between an artefact in an object-oriented programming language such as Java, and an artefact in SQL. The artefacts of each language reflect the conceptual framework on which it is based.

An object-oriented language is one interpretation of object concepts. In the Java language a class can have a single parent class and so form part of a hierarchy of classes. The semantics of a Java class are different from a C++ class in this respect (Weber 1997), p1084. A C++ class can have more than one parent class. Consequently whilst the term class is used at the concept and language levels it does not mean that in each case the term refers to the same thing or that within the language level two interpretations of the term are fungible.

SQL is one interpretation of relational concepts. A table is a representation of a relation and a row is a representation of a tuple. However the semantics of a relational tuple are not the same as those of an SQL row. For example a tuple is unique by definition in the relational model, whereas a duplicate row is allowed by the SQL standard definition and uniqueness must be imposed using a primary key.

Another difference between an object-oriented programming language such as Java and SQL:1992 is the extensibility of their type systems. Whereas a class is part of the Java type system and may have an arbitrary structure, there is no equivalent in the SQL:1992 type system. In SQL:1992 a data type is part of the language and its definition cannot be changed.

SQL provides an approximation of the data structures prescribed by the relational framework, just as Java provides an approximation of those prescribed by the object framework. The syntax and grammar of SQL is defined by standard and is implemented in vendor-specific languages such as Oracle and Sybase. None of these vendor-specific languages is a pure implementation of SQL but nevertheless can be classified as a relational language. Consequently each implementation of SQL offers a different choice of artefacts for the description of a relational database schema.

Each difference between artefacts in two languages materialises as a problem of an object-relational impedance mismatch. This kind of object-relational impedance mismatch is labelled a representation mismatch. A pattern dialogue is concerned with a representation mismatch.

A pattern dialogue will explore a difference between two artefacts in separate silos at the language level of the framework. For example it is possible to make a correspondence between a Java class and a SQL:1992 table and an attribute of a class and a column, but such a correspondence is partial. An attribute of a class can adopt a

programmer-defined class as its data type but a column can only adopt a language-defined type. Furthermore a column can only store the value and not the behaviour of an object. Consequently some of the semantics of a class can be lost when such a mapping strategy is used.

4.3.3 Schema Level

A dialogue at the schema level is concerned with understanding differences between two representations of some thing from a universe of discourse in separate silos of the framework. The emphasis here is on design issues. That is designing appropriate schemas for a database and a program to satisfy the requirements of some application.

The description of some thing from a universe of discourse within an object-relational application will involve at least two schemas: one based on a class and the other based on a table. Each schema is the product of a transformation process such as that described in Chapter 2.

Two representations of some thing from a universe of discourse are different not just because they are phrased in a different language, but because the purpose and priority of those responsible for an object-oriented program are different from those responsible for a relational database schema. Those who design an object-oriented program will focus on such things as coupling (Ambler 2003), p79 and a cohesive representation of a network of interacting objects. The concerns of those who design

a relational database include such things as data consistency, access performance, space used and the availability and security of data (Date 1986), p12-15.

Each difference of purpose and priority produces a kind of object-relational impedance mismatch that is labelled an emphasis mismatch. A failure to adequately address an emphasis mismatch will result in a loss of semantics between an object-oriented program and a relational database. A mapping dialogue is concerned with an emphasis mismatch.

A mapping dialogue could be used to understand the consequences of, for example, a Synthetic Object Identity (Keller 1997) or shadow information (Ambler 2003), p228 introduced in Chapter 2. Each is a choice of design at the schema level that addresses a mismatch. The objective of a mapping dialogue is to understand why they are necessary and any compromise that must be made.

4.3.4 Instance Level

The instance level of the framework is concerned with those artefacts present in an object-relational application at run-time.

The schema of an object-oriented program and the schema of a relational database define the storage structures into which data must fit. Issues relating to the treatment of data arise because, at the schema level, some of the semantics of an object are not preserved in a relational database. One such issue is the treatment of an object.

The problem is that an object is conceptualised as an atomic unit when in practice it has a number of subdivisions. A Java object has subdivisions of structure, value and behaviour. Figure 14 demonstrates the subdivisions across the schema and instance levels of the framework. In an object-relational application in order to understand the semantics of an object it is necessary to understand these subdivisions.

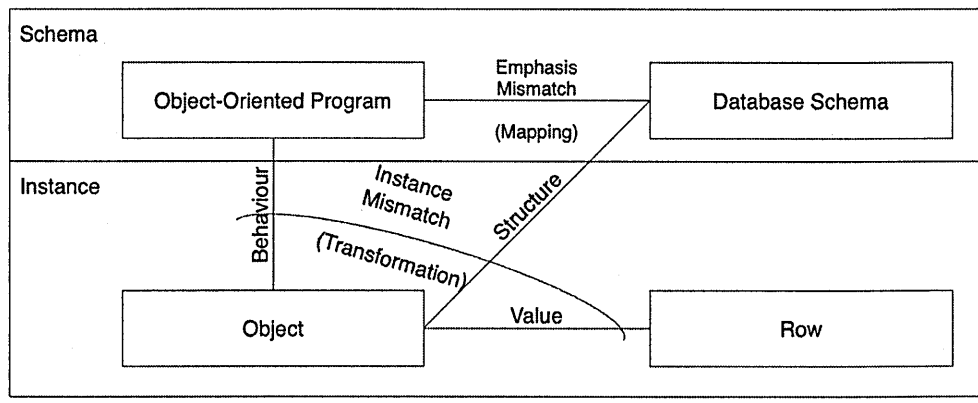


Figure 14 - Fragmentation of the Subdivisions of an Object

In order to understand the behaviour of an object it is necessary to refer to both the data of an object and the schema of a program. The structure of data is defined both in a Java class and an SQL schema. Functions and procedures were added to the SQL:1992 standard although relational database vendors have provided such facilities for some years. The valid state of an object can be enforced by a rule defined within a class method or as a database constraint. A data value may have a different format in a program and a database.

Such a fragmentation is characteristic of a difference at the instance level. This kind of object-relational impedance mismatch is labelled an instance mismatch. A transformation dialogue is concerned with an instance mismatch.

A transformation dialogue will explore the consequences of fragmentation. For example if the state of an object is to be stored in a database then the semantics of a class method and a database constraint must not conflict. There must also be a consistent interpretation of a value.

4.3.5 Reflection

This section reflects on the relevance, structure and possibilities of the framework. Chapter 3 concluded that relationships between problem themes cannot be used to identify the cause of a mismatch but that themes of schema ownership, structure and identity are important. The framework is an abstraction of an object-relational application based on the concerns of Copeland & Maier, Ambler and Neward, and the levels of the framework can be used to explore the cause of a mismatch.

All levels of the framework are concerned with an object-relational application but the schema and instance levels describe a particular application. So whilst a mismatch of structure or of identity can be observed at the schema or instance levels of the framework, all the levels of the framework can be used to explore the cause of that mismatch. A mismatch can be caused by a choice of a design at the schema level, a

choice of a language at the language level, or a choice of perspective at the concept level.

The framework is relevant to those who develop an object-relational application. At each level of the framework the concern is with an artefact involved in an object-relational application. At the schema level the concern is with artefacts in the schema of an object-oriented program and the schema of a relational database. At the language level is a language of implementation such as Java, C++ and SQL. At the instance level the concern is with the treatment of an object and its value. The concept level represents a concern for two ways of understanding the world.

The schema ownership problem is reflected in the silos of the framework. Each silo represents a particular concern across each level of the framework. Between silos there is a separation of concerns characteristic of the schema ownership problem. Chapter 2 identified some of the concerns of those responsible for an object-oriented program and those responsible for a relational database. Such concerns are those at the schema level of the framework. The framework also recognises the separate concerns of those responsible for the semantics of a language and a conceptual framework.

Ambler (Ambler 2003), p111 describes a cultural impedance mismatch. The framework, through the use of common levels of abstraction, facilitates a discourse between proponents of object and relational technologies. However, whereas Ambler

is concerned with cultural issues surrounding the development of a schema, the framework also includes such issues that arise at the language and concept level. A mismatch can now be addressed in a structured and consistent way, not just across levels of the framework but also between the object and the relational silos.

The language of the classification provides a new precision by which a mapping strategy can be understood. There are different kinds of object-relational impedance mismatch. Understanding a mismatch is the concern of a specific dialogue. A dialogue involves a different collection of artefacts at a particular level of the framework. Consequently a dialogue supports an understanding of the fidelity and integrity of both current and future mappings.

A particular group of stakeholders would be involved in a dialogue. At the schema level, those involved in the design and implementation of an object-relational application would address the different schemas and technologies. At the language level standards bodies and those responsible for the production of programming and database languages would exchange ideas about data and processing structures. At the concept level research bodies and those concerned with the way the world is conceived would try to reconcile their different perspectives.

Because of the context between levels, a dialogue can have a consequence at another level of the framework. For example a dialogue can result in a change to a language and that change could provide a new artefact for the design of a schema. However

each dialogue must only deal with a mismatch at the corresponding level of the framework. A mapping dialogue for example must only address a mismatch relating to the representation of some thing from a universe of discourse in a schema. A pattern dialogue must only address a mismatch relating to a correspondence between artefacts of two languages.

The framework promotes thinking in an integrated way. It is possible to exploit the strategic options available when defining a mapping strategy. For example by engaging in a pattern dialogue it is possible to understand how decisions made in the design of SQL correspond to structures in the Java language.

The framework can be used to work toward an appropriate solution in a structured way. In order to successfully resolve a mismatch it may be necessary to engage in a dialogue at more than one level of the framework. As a result an appropriate solution can involve a change to artefacts or a change to a correspondence at more than one level of the framework. This is not the same as a mapping strategy that considers object or relational issues in isolation such as Ambler (Ambler 2005) and Fussell (Fussell 1997). Such a mapping strategy lacks the clarity and scope provided by the framework.

The same term but with different interpretations can be used across the levels of the framework. Chapter 2 observes that the use of a single term "object" both in a conceptual object model and in the schema of an object-oriented program provides

an illusion of continuity. That illusion can be exposed across the levels of the framework. The term "class" is used at the concept, language and schema levels. However the semantics of a Java class are different from those of a C++ class. Each is a particular interpretation of the concept of a class so the term class should not be used without also making clear the level of the framework that applies.

There is a distinction between different kinds of object-relational impedance mismatch. The framework is an abstraction over an object-relational application. The categorisations of Ambler and Neward are based on observations of an object-relational application. So when Ambler and Neward use the term object-relational impedance mismatch they might be referring to an instance mismatch, an emphasis mismatch, a representation mismatch or a conceptual mismatch.

The literature has not explicitly recognised that it is possible to think of object-relational impedance mismatch at different levels of abstraction. At each level of the framework there is a particular kind of object-relational impedance mismatch that is addressed using a particular dialogue. As a result it is not certain that a technology such as Hibernate (Hibernate), or changes to the SQL standard (Eisenberg and Melton 1999) adequately addresses a mismatch and in an appropriate way. The framework and classification provide a means to understand the potential and consequences of these technologies; in the case of Hibernate because it employs one or more mappings, and in the case of changes to the SQL standard because it involves a change at the language level of the framework.

Through the medium of the framework there is now a way to understand the scope and consequences of a solution. The relationship of context between the levels of the framework can be used to explore the consequences, for example, of a change to the SQL standard. At each level a dialogue will establish the efficacy of a solution.

4.3.6 Summary

The framework is based on four levels of abstraction over an object-relational application. Silos representing the separate concerns of object and relational technologies cross all four levels. At each level it is possible to explore a difference between artefacts across silos. Each difference between silos is characteristic of a particular kind of object-relational impedance mismatch. Each mismatch is addressed by a different kind of dialogue:

- A conceptual mismatch provides the impetus for a reconciliation dialogue at the concept level;
- A representation mismatch provides the impetus for a pattern dialogue at the language level;
- An emphasis mismatch provides the impetus for a mapping dialogue at the schema level; and
- An instance mismatch provides the impetus for a transformation dialogue at the instance level.

The concerns of each kind of dialogue are summarised in Table 4.

Table 4 - The Concerns of a Dialogue at Each Level of The Framework

Level	A Dialogue is Concerned With...
Concept	Issues relating to an incompatibility between two different views of the world: one as a network of interacting objects and the other as a collection of relations.
Language	Issues relating to an incompatibility of artefacts between an object and a relational language.
Schema	Issues relating to the maintenance of two representations of some thing from a universe of discourse described in different languages.
Instance	Issues relating to semantics in the context of an object-relational application.

Copeland & Maier, Neward and Ambler base their categorisations of an impedance mismatch on observations of the way a program and a relational database interact. Typically such a mismatch materialises at the schema or the instance level of the framework because those levels represent respectively artefacts of the design and the execution of an object-relational application. However all the levels of the framework can be used to understand at which level a mismatch is appropriately addressed.

At each level of the framework it is possible to explore a correspondence between artefacts in each silo, both in terms of the semantics of each artefact and the basis of a correspondence. Such an exploration when taken across levels facilitates an understanding of the cause of a mismatch and the consequences of a solution.

4.4 Precision in the Definition of a Mapping

This section and the next demonstrate how the framework is used to improve the fidelity and integrity of a mapping strategy. The definition of a mapping strategy must take account of a level of the framework. For example, the statement "a class

corresponds to a table" is not safe and can be interpreted in a different way at each level of the framework.

At the language level the statement above can be interpreted as "a Java class corresponds to an SQL:1992 table" or "a C++ class corresponds to an SQL:1992 table". At the schema level the statement can be interpreted as, for example, "the class Order corresponds to the table ORDER" or "the class Instrument corresponds to the table STOCK". At the concept level such a statement is meaningless because there is no concept of a table at that level.

In the interests of precision, each such definition must make clear the level of the framework that applies. This can be achieved by making explicit the dialogue involved. In the case of the language level it is a pattern dialogue. In the case of the schema level it is a mapping dialogue.

In order that it is clear and unambiguous, the definition of a mapping strategy should not cross over levels of abstraction in the framework. It is not safe to say for example, that a class called "Order" corresponds to an SQL:1992 table. Figure 15 demonstrates that such a statement crosses levels of abstraction. For reasons of clarity and ambiguity it is necessary to specify to which table in a database schema the class Order corresponds.

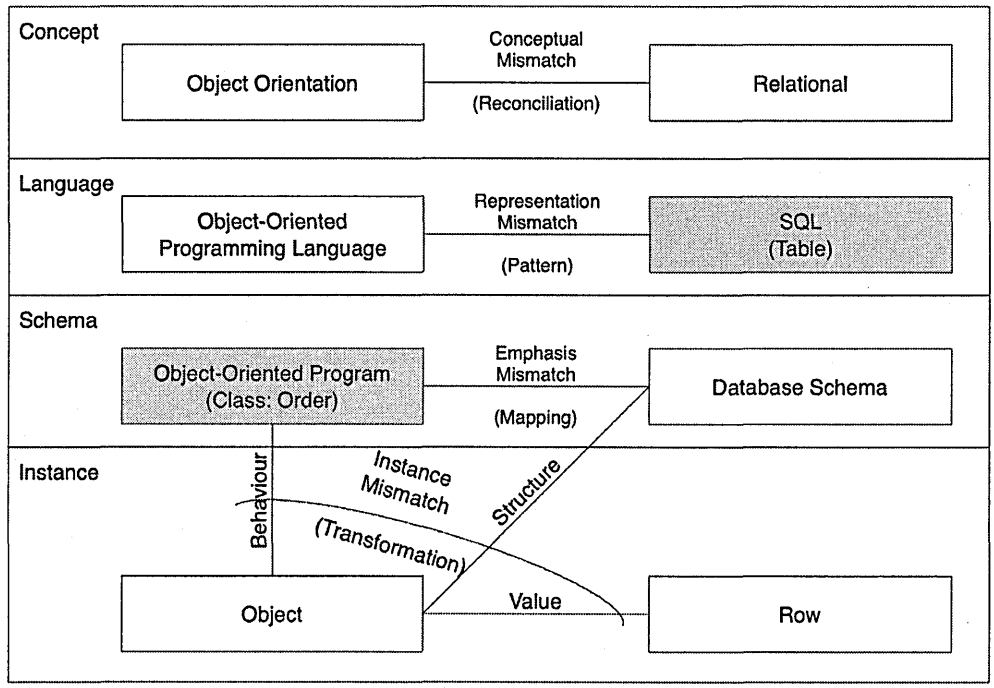


Figure 15 - Mixing Levels of Abstraction

The framework can be used to improve the definition of an existing mapping strategy. When referring to the mapping of a hierarchy of classes Ambler (Ambler 2003), p234 states, "There are tables corresponding to each of the Customer and Employee classes...". The precision of this statement can be improved thus "there is an SQL:1992 table called CUSTOMER with which there is a mapping to a Java class called Customer and there is an SQL:1992 table called EMPLOYEE with which there is a mapping to a Java class called Employee". Whilst terms from both the language and schema levels are used they are combined in a consistent way across silos.

4.5 *The Use of an Appropriate Language*

Effective communication between those responsible for an object-oriented program and those responsible for a relational database dictates that a specific set of terms is employed at each level of the framework. This section presents some initial guidelines.

When engaging in a dialogue a consistent use of a term will help to communicate accurately both a mismatch and a correspondence. In the previous section the statement "a class corresponds to a table" is ambiguous because both a class and a table can be considered an artefact of a language and an artefact of a schema.

Table 5 - Guidelines for Terminology at each Level of the Framework

Level	Terminology Guidelines
Conceptual	Terms relating to a particular conceptual framework, irrespective of how it is actually described or implemented. Example terms include class, object, relation, tuple and union.
Language	Terms relating to a language semantics, syntax and grammar, irrespective of a design. Example terms include Java Class, C++ Class, SQL:1992 table and column.
Schema	Terms relating to a specific schema. Example terms from the case study in Appendix B include Order, Trade and Equity.
Instance	Terms relating to data semantics. Example terms include instance, row, value, format and cast.

A mapping strategy must be phrased using a term in each silo at an appropriate level of the framework. It is possible to interpret the statement "a class corresponds to a table" in two ways but the suggested rephrasing in section 4.4 makes the semantics of a correspondence clear. This is because the rephrasing makes use of a consistent terminology at an appropriate level of the framework.

Table 5 suggests some guidelines for terms at each level of the framework. These terms are just an example but a starting point. The use of an appropriate set of terms will help to make a dialogue both specific and meaningful.

4.6 Fussells Classification of Object-Relational Mapping

Fussells classification (Fussell 1997), p20 of object-relational mapping comprises five "levels of object-relational mapping sophistication": (1) purely relational; (2) light object mapping; (3) medium object mapping; (4) full object mapping; and (5) object server. It is possible to think of each of Fussells levels as occupying a point along a continuum.

Using a "purely relational" approach a programmer works in terms of a relation both in a program and in a database. Using an approach based on an "object server" both the schema of a program and the schema of a database are based on the concept of an object. At these extremes of the continuum a programmer is working with artefacts of a single conceptual framework so an object-relational mapping is not necessary. It is only when a programmer must combine object and relational technologies and therefore move away from the extremes that an object-relational mapping is necessary. The relevant portion of Fussells classification is summarised in Table 6.

Table 6 - Fussells Classification of Object-Relational Mapping Sophistication

Classification	Fussells Description
Light Object Mapping	Try to isolate most of the application code from the specifics of SQL. Encapsulate SQL code in certain classes.
Medium Object Mapping	Primarily use objects and write all application behaviour in terms of objects. Convert a row from the database into an object and work on that object. Specify retrieval in terms of an object model.
Full Object Mapping	Completely use objects in the application code. Combine multiple object queries into larger database queries.

Fussell (Fussell 1997), p20 identifies drivers that can influence movement along the continuum from a relational-based to an object-based approach. He identifies two drivers as application size and complexity but defines neither. He claims that the management of a large and complex application is made somehow easier using an object-based approach. It is a choice for each project whether a light, medium or full object mapping is used.

4.6.1 A Comparison with The Framework

Fussells notion of sophistication is based on the extent to which a programmer of an object-oriented program is exposed to the schema of a relational database. Sophistication therefore relates not to an object-relational mapping strategy but to the cohesiveness of an object-relational application. In other words, for Fussell the sophistication of an object-relational mapping is concerned not with why a mapping is necessary but where in an object-relational application such a mapping can be implemented. Consequently Fussells classification of object-relational mapping

sophistication is orthogonal to the framework, reflecting issues of implementation rather than the cause of a mismatch.

In order to understand the cause of a mismatch it does not matter where in an object-relational application an object-relational mapping strategy is implemented because a mismatch occurs when artefacts from separate schemas are brought together. The issues of concern when trying to identify the cause of a mismatch are on what correspondence a mapping strategy is based, whether that correspondence involves a compromise and the reason for that compromise. Where in an object-relational application a mapping strategy is implemented is a secondary issue. However such an issue is of concern to those who develop an object-relational application.

Fussell observes that issues of an object-relational mapping are separate from those of a client/server architecture, but notes that such issues are intrinsically bound. The framework is not directly concerned with issues of client/server architecture but such issues do relate to the instance problem because they involve the canonical copy of state. However there is a mismatch not because there is a client and a server but because there are differences between artefacts. The framework recognises a separation of artefacts in two silos but neither silo necessarily represents a client or a server.

4.7 Summary

A four-level framework is presented in support of the hypothesis presented in section 1.3. At each level of the framework a particular kind of object-relational impedance mismatch is identified; understanding each mismatch is the concern of a dialogue. Reflecting on the framework and the classification:

- All the levels of the framework can be used to identify the cause of a mismatch;
- The cause of a mismatch may be at any level of the framework;
- Using the framework it is possible to work towards an appropriate solution in a structured way;
- A dialogue involves an exchange of ideas, between stakeholders, across silos at each level of the framework;
- The use of an appropriate language at each level of the framework helps to make a dialogue both specific and meaningful;
- The same term, but with different interpretations, can be used across the levels of the framework;
- The language of the classification provides a new precision by which a mapping strategy may be understood; and
- The framework provides a means to understand the potential and consequences of a solution.

Across the levels of the framework it is possible to explore the cause of a mismatch. Of concern is not for where in an object-relational application a mapping strategy can be implemented but rather why a mapping strategy is necessary. The issues of concern

when trying to identify the cause of a mismatch are on what correspondence a mapping strategy is based, whether that correspondence involves a compromise and the reason for that compromise.

Improving the definition of a mapping strategy began to demonstrate the efficacy of the framework. However before it is possible to identify the cause of a mismatch it is necessary to understand the nature of a compromise between artefacts at each level of the framework. An exploration of the nature of a compromise between artefacts provides the motivation for the next chapter.

Chapter 5 Understanding a Mismatch

5.1 Introduction

This chapter demonstrates that a mismatch, at the schema level, occurs not just because of a difference of language or of abstraction, but because the semantics of an object and a relational artefact are not equivalent. A technique is developed, in the context of the framework, for exploring a mismatch based on a notion of equivalence (Ireland, Bowers et al. 2011).

Equivalence is concerned with the semantics of a universe of discourse and how these are represented in the schema of an object-oriented program and the schema of a relational database. It is these semantics that must be preserved in a *round-trip* between an object-oriented program and a relational database.

A mapping strategy embodies a correspondence between two schemas but typically involves a compromise. An equivalence diagram is used, in support of a mapping dialogue, to explore a mapping strategy and to understand that compromise. Such an analysis illuminates a mismatch as a first step toward understanding the cause of a mismatch.

The notion of equivalence is demonstrated using an example based on the concept of identity. In terms of an identity, there is very little in common between an object and a relational artefact. Consequently a mapping strategy such as that described by Blaha

(Blaha, Premerlani et al. 1988), p420 is based on a false correspondence. The levels of the framework are used to explore the cause of this mismatch of identity.

Using an equivalence diagram it is possible to explore latent issues with a mapping strategy. A Synthetic Object Identity (Keller 1997),p21 is an attempt to provide a consistent notion of an identity between the schema of an object-oriented program and the schema of a relational database. However it is possible to interpret a Synthetic Object Identity in different ways. Demonstrated are both the need for a consistent interpretation of a Synthetic Object Identity between the silos of the framework, and the consequences for data integrity of a different interpretation.

The notion of equivalence has consequences for the framework and how it is possible to think about a mapping strategy. A third silo in the framework is proposed. The reference silo is concerned with the description of a universe of discourse in a way that is independent of a description in both the schema of an object-oriented program and the schema of a relational database. Whilst the reference silo is currently theoretical, there is existing work that might provide a basis for the description of a universe of discourse. Artefacts in the reference silo can be used to explore equivalence and to inform a dialogue at other levels of the framework.

5.2 A Round-trip

The schema level of the framework relates directly to the work of those involved in the production of an object-relational application. At this level the concern is with

artefacts that comprise respectively the schema of an object-oriented program and the schema of a relational database. If an object-oriented program is to use a relational database for storage then data must pass between them.

A *round-trip* is distinct from a mapping strategy. In a round-trip data about an object is stored in a database and, save for its identity, that same object can be re-created at a later time. A round-trip will involve two mapping strategies: one from a program to a database and another from a database to a program. A mapping strategy embodies a correspondence between the schema of an object-oriented program and the schema of a relational database. This chapter is concerned with the extent to which mapping strategies support a round-trip.

Both the schema of an object-oriented program and the schema of a relational database include a description of the same entity because they are part of the same object-relational application. An entity is any concrete or abstract thing of interest (Griethuysen 1982), p2-1 from a universe of discourse. Each schema includes a description of an entity for the purpose of an implementation. So whilst each schema is a different abstraction, both schemas include a partial representation of the same entity.

An entity provides a point of reference common to both the schema of an object-oriented program and the schema of a relational database. If the semantics of an entity are not to be lost in a round-trip between an object-oriented program and a

relational database then these schemata must include equivalent descriptions of that entity.

5.3 Two Representations of An Entity

Figure 16 shows an object and a relational representation of the same entity (or entity type) at the schema level of the framework. In this example the schema of an object-oriented program is formed using artefacts from the Java language. The relational database schema is formed using artefacts from SQL:1992. The assumption is made that each schema includes as complete a representation of the semantics of an entity as possible within the constraints of a given language in a silo of the framework.

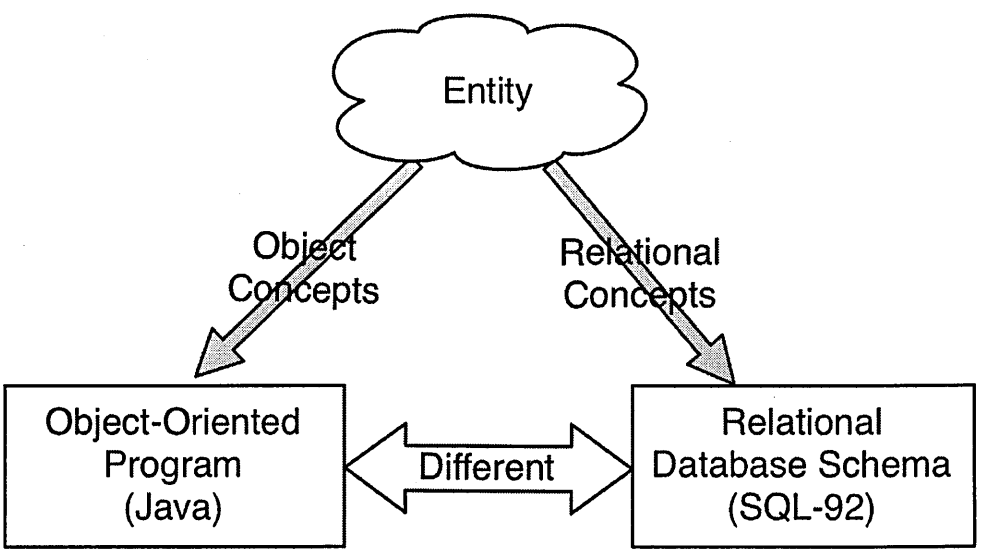


Figure 16 - Two Representations of an Entity (Type) at the Schema Level

The semantics of an entity provide a common point of reference for both an object and a relational schema. An entity can also be understood as a generalisation of an

object and a relational representation. An entity represents the semantics of some thing from a universe of discourse that can be used as a baseline to compare an object and a relational representation. Such a comparison is necessary in order to understand a correspondence and any compromise that is necessary to enable that correspondence.

5.4 Equivalence

Two representations are considered to be equivalent, at the schema level, if they each describe the same semantics of an entity from a universe of discourse. Only those semantics that are equivalent can form part of a non-loss round-trip between an object-oriented program and a relational database.

If semantics of an entity are completely described in both an object and a relational schema, then none of the semantics of that entity should be lost in a round-trip between an object-oriented program and a relational database. There may still be differences, for example in structure, but these would not impact on the semantics captured by the different representations.

Where there are semantics that cannot be preserved in such a round-trip, then one schema is able to describe more (or a different subset) of the semantics of an entity than the other. Each such difference is a mismatch. The expectation is that a dialogue, about a mismatch, will be informed by an analysis based on this notion of equivalence.

In both an object and a relational schema one or more language artefacts can be used to describe an entity. An equivalence diagram is used to explore differences in the semantics of an entity as represented by these artefacts. An equivalence diagram embodies the notion of an equivalence and focuses attention on the essential aspect of Figure 16: that each schema includes a description of the same entity.

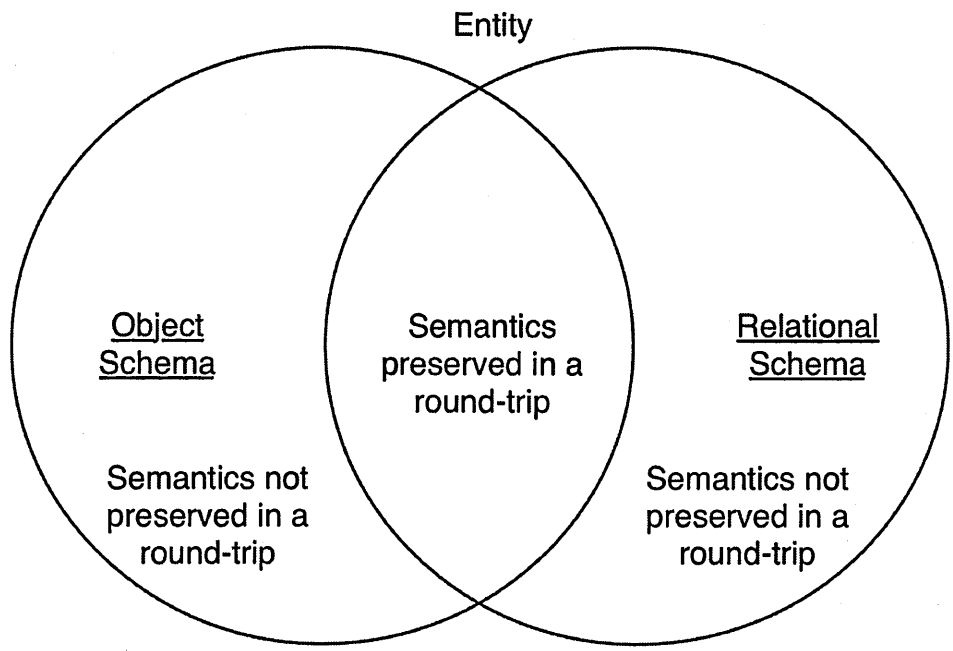


Figure 17 - Equivalent Representations of an Entity at the Schema Level

An equivalence diagram is a Venn diagram comprising two sets. Each set contains a description of the semantics of an entity in a particular schema. The intersection of these two sets is those semantics of an entity that are captured in both schemas. These semantics will be preserved in a round-trip between an object-oriented program and a relational database. Those semantics in each set that lie outside the intersection will not be preserved in a round-trip.

In Figure 17 the semantics of an entity embodied in artefacts used in a schema are represented by a set, drawn as an ellipse. Two sets are shown: an object schema and a relational schema. The intersection of the two sets represents the semantics of an entity common to both schemas. These semantics need not be expressed in the same way but they are semantically equivalent descriptions of that entity.

The following sections demonstrate the use of an equivalence diagram to understand a correspondence. First, in section 5.5, an equivalence diagram is used to explore a correspondence of identity between two representations of an entity. Second, in section 5.6, an equivalence diagram is used to explore latent issues with a concept of identity introduced in a mapping strategy.

5.5 Equivalent Identities

This section demonstrates the use of an equivalence diagram to explore a correspondence based on the problem of identity described in Chapter 3. That problem, at the schema level, is how to uniquely identify a representation of an entity across both an object-oriented program and a relational database.

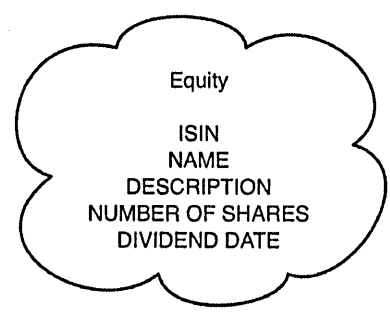


Figure 18 - The Entity Equity

Figure 18 presents an entity "Equity" taken from the universe of discourse described in the FTI Case Study in Appendix B. Equity is a particular kind of financial instrument that represents a share in a company. The value of an International Securities Identifying Number (ISIN) provides an identity for a particular equity.

Because two things are distinct means there is some way to separate them. An entity identifier is a means by which an entity in a universe of discourse is distinguished. In this example ISIN is an entity identifier for the entity Equity.

```
… class Equity
{       … ISIN;
        … NAME;
        … DESCRIPTION;
        … NUMBER_OF_SHARES;
        … DIVIDEND_DATE;   }
```

Figure 19 - An Outline of a Java Class Equity Derived from the Entity Equity

An ISIN is defined under ISO 6166 and is unique across all financial instruments. The other attributes are described in Appendix B. From Figure 18 is produced an outline class definition shown in Figure 19 and an outline SQL:1992 table definition in Figure 20.

```
create table EQUITY(
   ISIN … PRIMARY KEY,
   NAME …,
   DESCRIPTION …,
   NUMBER_OF_SHARES …,
   DIVIDEND_DATE …,)
```

Figure 20 - An Outline of a SQL:1992 Table Derived from the Entity Equity

5.5.1 The Identity of an Object

At the schema level an object identity ("object ID") is implicit and is the identity of an object. Using the Java language it is not necessary to include an object ID in the definition of a class. Hence, there is no mention of an object ID in the definition of class Equity in Figure 19.

The value of an object ID is independent of the value of any of the attributes of an object. A programmer does not choose, or influence the value of an object ID. Although an ISIN is unique, the value of an object ID of an object of class Equity is completely independent of the value of the attribute ISIN.

An object ID is guaranteed to be unique within the execution space of an object-oriented program. Two objects with exactly the same attribute values are different objects if they have a different object ID. Consequently two objects of class Equity are different even if they have the same value for the attribute ISIN.

The identity of an object remains the same regardless of any changes to the value of its attributes. The same object can be used, over time, to hold data about different entities. Consequently, in the case of an object of class Equity, changing the value of the attribute ISIN does not change the value of its object ID.

5.5.2 The Identity of a Row

At the concept level of the framework the identity of a tuple is the value of all its domains. As such the identity of a tuple is dependent on the value of these domains. In a single relation there cannot be two tuples with the same value for each domain. Codd (Codd 1970), p380 introduced the idea of a primary key for a relation but it is not necessary for identity. However such a primary key does provide a short-form of reference to a tuple.

At the language level, the semantics of a table are different from those of a relation at the concept level. A duplicate row is permissible. A SQL primary key enforces the uniqueness of a row in a table and restricts the identity of a row to those columns in that key. For example in Figure 20 the column ISIN has been chosen as the primary key of the table EQUITY. There cannot be two rows in this table with the same value for this column. The value of a primary key should not be changed because that affects the identity of a row.

5.5.3 The Semantics of Identity Compared

Figure 21 is an equivalence diagram for the semantics of the identity of entity Equity. This demonstrates that there is little in common between the object and relational semantics of identity. A row and an object are not the same thing. An object ID and the primary key ISIN have little correspondence. The only semantics they share is that each uses identity as a mechanism for ensuring an occurrence is distinct.

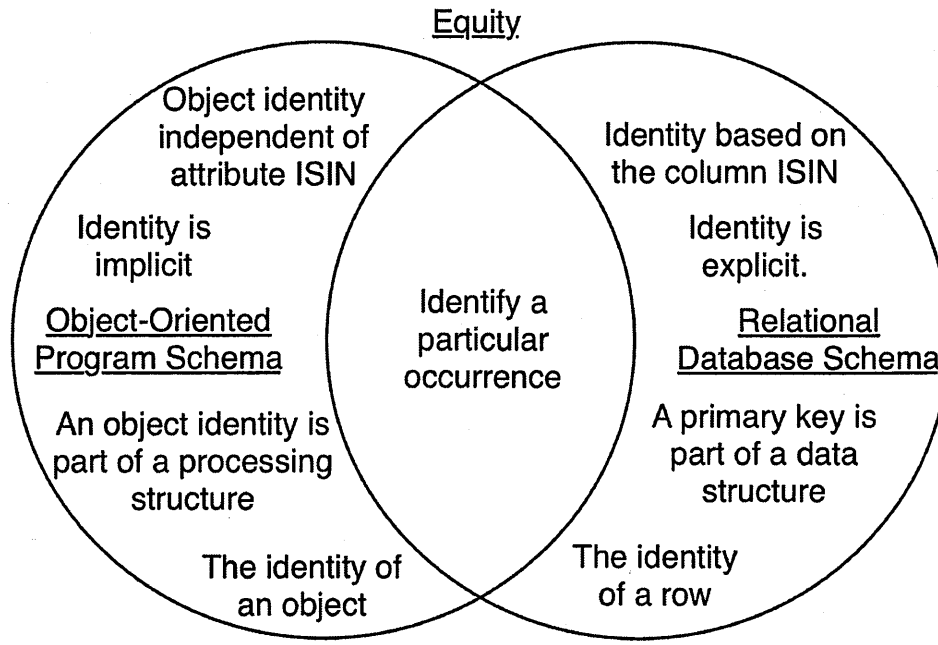


Figure 21 - Equivalent Semantics of Identity for Representations of the Entity Equity

Blaha (Blaha, Premerlani et al. 1988), p420 makes a correspondence between an object ID and a primary key. Even in this simple example this mapping strategy has shortcomings. An object ID is unique only within the execution space of a single object-oriented program. Consequently the correspondence between an ISIN and a specific object ID is only temporary.

An object ID does not have the semantics necessary to be used as a primary key in a relational database schema. The value of an object ID should not be used as the value of a primary key because it is not guaranteed to be unique in a database. Even if the value of an object ID could be extracted, that value has no meaning in a database. It also has no semantics in the universe of discourse of which a database schema is a representation.

The levels of the framework can be used to pinpoint the cause of this mismatch of identity. Differences between an object ID and a primary key are realised at the language level of the framework and above.

An object ID is not a building block of the object framework at the concept level, rather it is a programming necessity introduced at the language level. At the concept level an object is distinct so there is no need for an object ID. Similarly in SQL, at the language level, a primary key uniquely identifies a row in a table but at the concept level a tuple is distinct by definition.

At the concept level there is an object and a tuple, each is unique but they are different concepts and have no common basis for uniqueness. Consequently at the language level it is not appropriate to make a correspondence between an object ID and a primary key because they have no common basis for uniqueness.

At the schema level each representation has the identity of an entity in common. In this example the attribute ISIN and the column ISIN serve to identify an entity. Consequently a correspondence should be made between the attribute ISIN (Figure 19) and the column ISIN (Figure 20) because these are equivalent forms of identity. They each refer to the same concept, that is, the entity Equity.

5.5.4 Reflection

Using the notion of equivalence it emerges that a correspondence of identity at the schema level should be based on correspondence between the mechanisms used to describe the identity of an entity. In the example a correspondence should be made between the representations of the attribute ISIN in each schema. This attribute provides an identity for the entity Equity and it is common to both representations.

The mechanism used to describe the identity of an entity may not be the same as the identity used in each identity system. The column ISIN is the primary key of table EQUITY but this is a choice of design. The attribute ISIN is separate from an object ID. Because a reference between objects is based on an object ID, a solution based on ISIN must be developed in an object-oriented program at the schema level.

What is important is that the identity of an entity is common to both representations. A correspondence between an object ID and a primary key should not be made because these identify different concepts and they have a different purpose. These conclusions are supported by practice. Ambler (Ambler 2003), p285 describes an object cache based on the Identity Map pattern of Fowler (Fowler, Rice et al. 2003). An object cache employs a mapping strategy based on a correspondence of entity identity.

5.6 Exploring the Concept of Identity

This section explores the concept of identity across all the levels of the framework. It is possible to interpret that to which an identity refers in two separate ways. The consequences of these different interpretations are demonstrated using an equivalence diagram whilst exploring latent issues with a mapping strategy.

5.6.1 Two Interpretations of an Object

In an object-relational application an object-oriented program will request data relating to an object from a relational database. The identification of the correct data is essential to the success of each such request. It is therefore important to be clear about that to which an identity refers.

In Appendix C the framework is used as a structure to understand some of the literature on identity in the context of an object-relational application. In the left column and at each level the concern is with the identity of an object. In the right column the concern is with the identity of a tuple (or row). In the centre column the concern is with issues of a correspondence between the two.

The analysis in Appendix C demonstrates that it is possible to think about the concept of an object in two different ways and at different levels of the framework. First, an object is a placeholder (or avatar) for an entity from a universe of discourse (Hall, Owlett et al. 1976). This perspective is adopted at the concept level and also at the schema level. At the concept level an object is a surrogate for some real-world thing.

At the schema level the design of an object-oriented program may be such that an object represents data about a particular entity.

Second an object is a container in a program for data about an entity from a universe of discourse (Wieringa and De Jonge 1991). This perspective is adopted at the language level and at the schema level. At the language level an object can be viewed as an address in memory that contains data about an entity. Over time that address may contain data about different entities. At the schema level the design of an object-oriented program may be such that, over time, an object contains data about a different entity. At the schema level of the framework therefore it is possible to interpret an object, and consequently that to which its identity refers, in two different ways.

The idea of a container is not new. Kent (Kent 1991), p12 refers to a form of object he calls a carrier (or intensional set). The identity of a carrier does not change even if its cargo does. Changing the value of an attribute (or all attributes) of a carrier does not change its identity. A carrier can then represent a different entity. An interpretation, based on the semantics of a container, has consequences for the semantics of a reference between objects and for the integrity of data.

Kent (Kent 1991), p2 states that "references need to be reliable, not affected by changes in things", but care must be taken with the semantics of a reference between two containers. Consider for example, two objects identified as A and B. Object A

references object B. Object B is a container. The data of object B is changed. It is not clear whether object A should still reference object B.

With regard to identity Kent (Kent 1991),p2 talks about durability (an object is still the same thing despite changes) and distinguishability (knowing whether there is one thing or two). Wieringa (Wieringa and De Jonge 1991),p3 talks about principles of persistence (an object identity remains invariant under any state of the object) and uniqueness (each object has one and only one identity which differs from the identity of any other relevant object). In the example above object A and object B are both durable and distinguishable and their identity does not change, but object B (as a container) no longer represents the same entity from a universe of discourse that it did before the change.

5.6.2 Two Interpretations of a Surrogate Key

The concept of a surrogate key was introduced in order to overcome problems with the use of an entity identifier as a primary key (Date 1985),p244. It is not always possible to select an identifier for an entity. There can be a choice of identifier for an entity. And unlike a surrogate key, the value of an entity identifier can be subject to change.

A surrogate key is used as the primary key of a table. A surrogate key is a choice of a design at the schema level of the framework. The value of a surrogate key is not meaningful outside a relational database (Date 1985),p245 . A database management

system typically generates the value of a surrogate key (Wieringa and De Jonge 1991),p4, (Brown and Whitenack 1994). The value of a surrogate key is not subject to change and must never be reused (Date 1985),p245.

Date (Date 1985),p245 states that "a surrogate key is permanently associated with an entity", but the use of a surrogate key does allow the value of a column that represents an entity identifier to change (Khoshfian and Copeland 1986), p413. It is a question of semantics as to whether such a modified row now represents a different entity or a modified entity. However such issues of semantics are important for data integrity.

It is possible to think about a surrogate key as the identity of two different abstractions. In the first a surrogate key is the identity of data about an entity stored in a row of a table. A column that represents an entity identifier is another way of locating a row. In the second a surrogate key is the identity of a row. That row is a container for data about an entity.

It is possible to interpret both an object and a surrogate key in two different ways. It is important for reasons of semantics and of integrity, to ensure that those who produce an object-oriented program and those who produce a relational database schema employ the same interpretation. The schema ownership problem theme demonstrates that this is by no means certain. The next section demonstrates the importance of a shared interpretation.

5.6.3 A Synthetic Object Identity

Keller (Keller 1997),p21 propose the use of a Synthetic Object Identity in a number of mapping strategies. A Synthetic Object Identity is used to make a correspondence of an identity between the schema of an object-oriented program and the schema of a relational database. A representation of the concept of a Synthetic Object Identity is included in both schemas. So whilst a Synthetic Object Identity is not an entity identifier, it is common to both representations and so might be an equivalent form of identity.

Unlike an object ID, a program rather than a run-time system assigns the value of a Synthetic Object Identity. Although it is not entirely clear how Keller would generate such a value but there are a number of options. For example JPOX (JPOX 2008) uses a timestamp or a function to generate a value, whilst Brown (Brown and Whitenack 1994) suggests using a database sequence number generator or column of a row of a table to hold the next available value. The value of a Synthetic Object Identity can be stored in a database as a surrogate key, outlive the execution of a program and be the same between executions of a program.

A Synthetic Object Identity provides an identity for an object. In an object-oriented program the identity of an object remains its object ID. In a relational database a Synthetic Object Identity is used as a surrogate key and so provides the identity for a row.

There is a choice of whether a value of a Synthetic Object Identity represents the identity of an entity or the identity of a container. Keller does not provide an interpretation for a Synthetic Object Identity. However it is important for reasons of integrity and that those responsible for the schema of an object-oriented program and those responsible for the schema of a relational database employ the same interpretation.

In Kellers mapping strategies a Synthetic Object Identity is used both as an attribute of an object and as a surrogate key, whilst it is possible to interpret a Synthetic Object Identity as the identity of an entity or an identity of a container. Consequently there are four possible combinations based on these two dimensions. The four combinations are considered in Table 7.

Table 7 - Understanding the Use of a Synthetic Object Identity

Object Attribute / Surrogate Key	As an identity of an object that represents data about an entity	As an identity of an object that represents a container of data about an entity
As the identity of a row that represents data about an entity	A correspondence is trivial. A value of a Synthetic Object Identity corresponds to a value of an entity identifier.	If the value of an object is changed, does that object still represent the same entity, a different entity (row) in a database, or a new entity?
As the identity of a row that represents a container of data about an entity	If the value of a row is changed, does that row still represent the same entity, a different entity (object) in a program, or a new entity?	The value of a Synthetic Object Identity may be the same in a program and in a database, but what are the consequences of making a correspondence between two containers?

Because a Synthetic Object Identity is an artefact of a schema, an equivalence diagram can be used to understand each combination. For simplicity the assumption is made that data about an object is stored in a row of a table, although from the structure problem theme it is evident that this is not always the case.

Each combination in Table 7 leads to a question of integrity between an object-oriented program and a relational database. In the next section, and by way of an example, the use of a Synthetic Object Identity as the identity of a container is explored. The objective is to answer the question in the bottom right cell of Table 7.

5.6.4 A Synthetic Object Identity as the Identity of a Container

In this section an equivalence diagram is used to explore latent issues with the use of a Synthetic Object Identity. An example is used, based on an Order for an Instrument, taken from the FTI Case Study in Appendix B. In this example a Synthetic Object Identity is an identity of a container in both schemas. The issues arising from the use of a Synthetic Object Identity in this way are summarised using the equivalence diagram Figure 22.

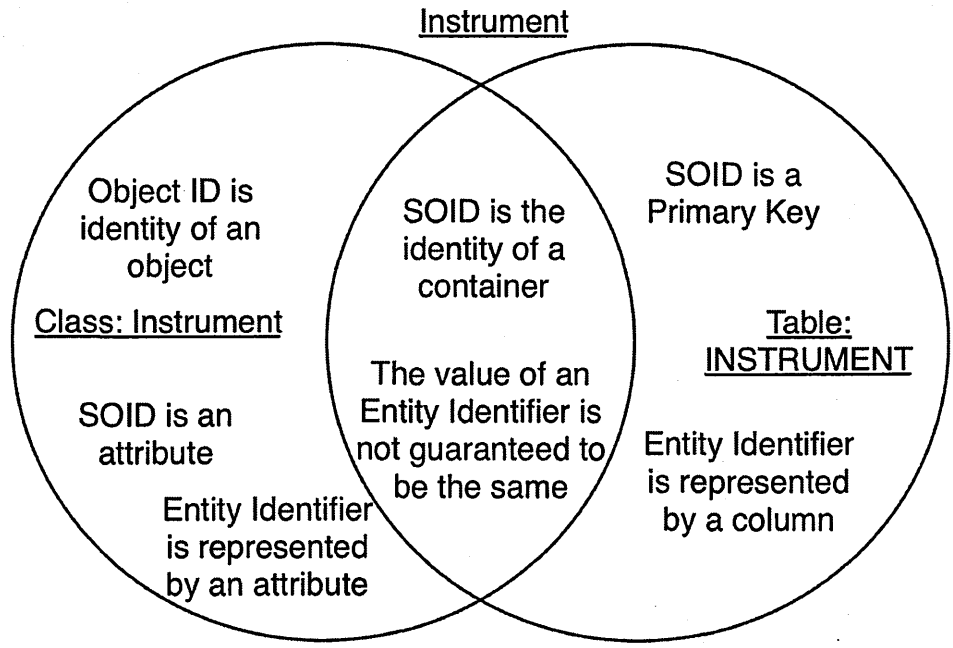


Figure 22 - A Synthetic Object ID (SOID) as the Identity of a Container

Consider an order on a financial market to purchase 1,000 Vodafone shares at 220p each. Table 8 shows an order created with a Synthetic Object Identity (Order SOID) of "1". The assumption is made that there is some way to generate this value and that it is sufficiently unique.

Table 8 - An Order to Buy 1,000 Vodafone Shares @ 220p

Time	Order SOID	Order Details	Instrument SOID	Instrument Entity ID	Name
1	1	Buy 1,000 Vodafone @ 220p	1	GB00B16GWD56	Vodafone
2	1	Buy 1,000 ICI @ 220p	1	GB0004594973	ICI

The order is entered at time "1" against an instrument with an entity identifier GB00B16GWD56. In the financial markets the entity identifier GB00B16GWD56 is an

ISIN that provides an identity for a share in the company Vodafone. The container holding data about this instrument has a Synthetic Object Identity (Instrument SOID) of "1".

At some time "2" data about a different entity is loaded into the container with Instrument SOID "1". The ISIN GB0004594973 identifies that entity as the company ICI. Because Instrument SOID provides an identity for a container the value of Instrument SOID does not change. However the semantics of the order have now changed. The details of the order at time "1" described a purchase of 1,000 Vodafone shares but now the data describes a purchase of 1,000 ICI shares.

This identity problem, typical of a surrogate identifier, can occur independently in a program, in a database or between a program and a database. The scenario described above could represent a valid change within the system but it is important, for the semantics of an order, that those involved each understands the semantics of a Synthetic Object Identity in the same way and the consequences of a particular interpretation.

5.6.5 Reflection

The notion of equivalence presented in this dissertation provides a way to explore latent issues and assumptions embodied in a mapping strategy. It is possible to conceive of both an object and a surrogate key as a placeholder for an entity or as a container for data about an entity. It is possible that a different interpretation is made

in each silo of the framework and this has a consequence for the concept of identity in an object-relational application.

Issues of data integrity mean that it is important to be clear about that to which an identifier refers. Such clarity is necessary not only in the schema of an object-oriented program and the schema of a relational database, but also in the definition of a mapping strategy between the two schemas.

Keller (Keller 1997), p21 proposes a Synthetic Object Identity in a number of mapping strategies. A Synthetic Object Identity provides an identity for an object and is used as a surrogate key of a table. Whilst an object ID and a primary key identify different concepts a Synthetic Object Identity identifies the same concept. However it is possible to interpret that concept in different ways.

Care must be taken to make a correct and consistent interpretation of the semantics of a Synthetic Object Identity. In the example of an Order and an Instrument (Table 8), a Synthetic Object Identity does not provide an identity for an entity instead it provides an identity for a container. A reference is between containers, not between representations of an entity, and the content of a container can be subject to change. Consequently such a change will not violate a foreign key constraint in a database nor will it affect a reference between two objects but it can change the semantics of that reference.

Issues with the interpretation of a Synthetic Object Identity are not insurmountable but cooperation is necessary. Those who produce a relational database schema may insist on the use of a surrogate key because of the benefits. A programmer may be reluctant to maintain a Synthetic Object Identity both because it is not necessary for a reference between two objects in a program and because an attribute representing the identity of an entity may be already part of the definition of an object. Each such tension can impede a solution and must be addressed by a mapping dialogue.

5.7 Describing an Entity

The language for describing an entity on an equivalence diagram is an important choice. So far the description of an entity is based on the case study in Appendix B. Such a description is sufficient in order to demonstrate equivalence. However the semantics of an entity will need to be described in a precise way if subtle differences in semantics are to be exposed.

A description of an entity cannot favour one of the two conceptual frameworks. This would limit the description of the semantics of an entity to those that can be expressed using just one of the frameworks. This section explores three possibilities for a language to describe an entity.

5.7.1 A Generic Conceptual Model

Dieste (Dieste, Genero et al. 2003) describe what they term a generic conceptual model. A generic conceptual model is independent of both an object and a relational

conceptual framework. An objective of a generic conceptual model is to describe knowledge of a requirement in a way that does not determine (what they refer to as) the implementation paradigm. They provide a number of transformations from a generic conceptual model to a model in a particular implementation paradigm. A paradigm underpins a conceptual framework and object and relational are two conceptual frameworks. A conceptual framework also underpins an implementation language.

The approach of Dieste is set within a process of software development such as that described in Chapter 2. The objective of a generic conceptual model is to delay commitment to an implementation model by providing a description of a universe of discourse independent of an implementation language. Once a choice of implementation language has been made, an element on a generic conceptual model is transformed into an artefact of a schema described using that implementation language.

The language employed in the production of a generic conceptual model is a possible candidate for the description of an entity. Unlike an element, an entity is not used as the basis for the generation of a schema. Rather a description of the semantics and data of an entity is used as the basis for exploring equivalence.

5.7.2 Multi-paradigm Modelling

Multi-paradigm Modelling is another area in which a candidate for the description of an entity may be found. Amaral (Amaral, Hardebolle et al. 2010) describes the concerns of Multi-paradigm Modelling as "developing a set of concepts and tools to address the challenge of integrating models of different aspects of a software system specified using different formalisms and eventually at different levels of abstraction". The schema of an object-oriented program and the schema of a relational database each reflect different formalisms.

Integrating heterogeneous models is an important challenge of Multi-paradigm Modelling. Amaral (Amaral, Hardebolle et al. 2010) notes that, "the topic on model composition is of very high interest but one that raises a number of very difficult issues". Various authors (Jiang et al., Yie et al. and Barroca et al. in (Amaral, Hardebolle et al. 2010), p222) have explored dependencies between models, model transformations and language composition. Because it must accommodate two models the language used for model integration might provide a way to describe an entity.

5.7.3 The Model Driven Architecture

Chapter 2 established that the Model Driven Architecture (MDA) will not avoid a mismatch, but the idea of a meta-model might prove useful for the description of an entity. The levels of the MDA are summarised in Figure 23.

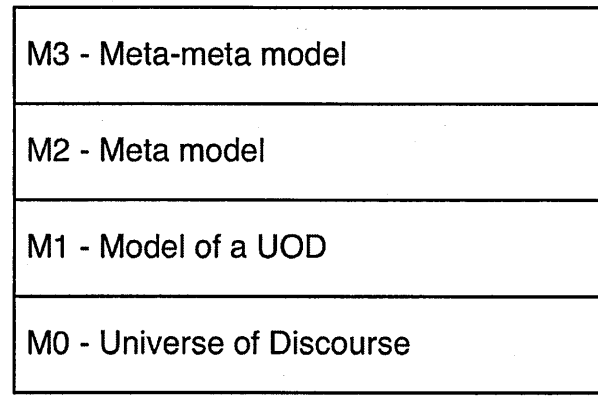


Figure 23 - The Levels of the Model Driven Architecture

Level M0 represent a universe of discourse. Level M1 is a model of a universe of discourse for a particular application. Level M2 is a model of the language used to describe a model at level M1. Level M3 is a model used to describe a language at level M2.

A parallel can be drawn between a schema at the schema level of the framework and a model at level M1 of the MDA. A language could then be defined at level M2 of the MDA for the description of an entity at level M1. A challenge is that such a language must not favour either the object or the relational conceptual framework. A description of an entity using this language could then be used to explore equivalence between an object and a relational representation of that entity at level M1.

5.7.4 Reflection

Because there is not yet a formal language for the description of an entity, the examples in this dissertation use the description supplied in the case study in

Appendix B. The objective is to demonstrate the outcomes possible using the technique of equivalence. The description in the case study is sufficient for this purpose.

There are other options for the description of an entity. The challenge in each case is to produce a description of an entity in such a way that does not favour either conceptual framework. That challenge is not addressed in this dissertation rather in the next section the consequences of equivalence for the framework are explored.

5.8 Equivalence and The Framework

The concept of an entity is only relevant at the schema level because a schema is a representation of a universe of discourse. This section explores the consequences of equivalence for all levels of the framework and explains the basis for equivalence at each level. At each level of the framework an equivalence diagram can be used to explore the different ways in which the semantics of an entity are described by artefacts across silos. Such an exploration of correspondence, when taken at each level of the framework, will inform a dialogue and illuminate a mismatch.

5.8.1 The Reference Silo

The concept level of the framework provides the context for the language level that in turn provides the context for the schema level. Such contextualisation can be used to reflect on a description of an entity at the schema level. For example, the identity problem explored in section 5.5 highlights that an issue at the language level, that is

that a primary key and an object ID provide the identity for different concepts, influences the semantics of an entity as described in a schema.

The representation of an entity can be viewed as a generalisation of an object and a relational representation. The description of an entity must be phrased in terms of a language that is itself a generalisation of an object and a relational language. A conceptual framework that is a generalisation of an object and a relational conceptual framework will underpin that language. Consequently a third silo in the framework is proposed and that silo is labelled the reference silo.

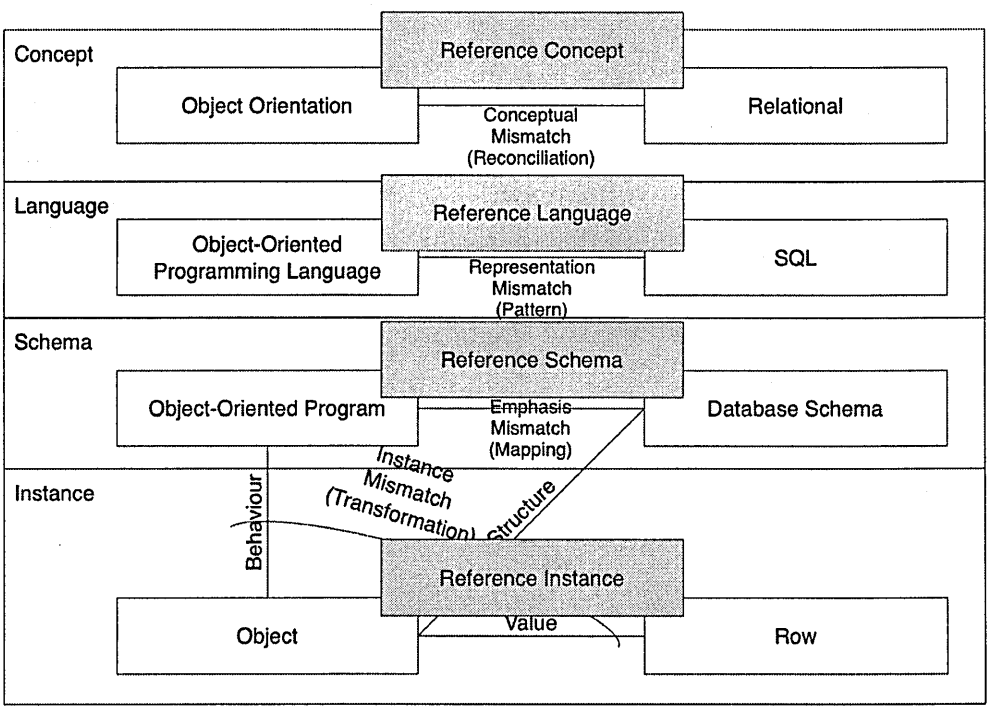


Figure 24 - The Framework Including the Reference Silo

The reference silo (shown down the centre of Figure 24) is currently theoretical and artefacts within it represent an ideal, but its purpose can be related to the work

described in section 5.7. Section 5.7 explored some of the options for the description of an entity and highlighted that the language used must be independent of both object and relational conceptual frameworks. Consequently such a language cannot form part of either the object or the relational silo. The purpose of the reference silo is to provide a way to describe an entity but in so doing the reference silo also provides artefacts for exploring equivalence at other levels of the framework.

In the reference silo there is a reference conceptual framework, a reference language, a reference schema and a reference instance. At each level the reference silo provides artefacts for, or influences the description of an entity from a universe of discourse within a reference schema. This description does not need to be perfect, but as a minimum it must be a generalisation of those semantics and data that can be described using an object and a relational artefact.

The reference silo is not software, nor is the reference silo a way to structure an object-relational application. At each level of the framework, and using artefacts from the reference silo, it is possible to explore equivalence and so inform a dialogue. It is possible to explore equivalence between the semantics of a reference artefact and those semantics described in an object and a relational artefact.

The semantics of a reference artefact described by an object or a relational artefact are shown as a set in an equivalence diagram. Depending on the level of the framework, that set can contain an artefact of a conceptual framework, an artefact of

a language, an artefact of a schema (such as an entity) or an artefact of data in an executing application. For example both Figure 21 and Figure 22 are equivalence diagrams that demonstrate the description of semantics at the schema level of the framework using the language taken from the FTI Case Study in Appendix B.

Table 9 - Some Building Blocks of the Object and the Relational Conceptual Frameworks

Conceptual Framework	Building Blocks
Object (Based on (Rumbaugh, Jacobson et al. 2005))	Object, Class, Association, Method, Attribute, Subclass
Relational (Based on (Codd 1970))	Relation, n-tuple, Domain, Column, Projection, Join, Restriction, Composition, Primary Key

At the concept level of the framework a set comprises the building blocks employed by a conceptual framework. Table 9 provides an example of some of the building blocks employed by the object and the relational conceptual frameworks. The intersection of two sets will comprise those semantics of a reference artefact that are represented by artefacts in both the object and the relational silo at the concept level of the framework. A difference between artefacts across silos is characteristic of a conceptual mismatch. The building blocks (Table 9) would form part of the language used in a reconciliation dialogue. That dialogue would be informed by an analysis based on an equivalence diagram.

An exploration based on equivalence at the other levels of the framework will facilitate an understanding of a mismatch. An exploration at a particular level will inform a dialogue between silos. Such a dialogue can result in a change to a mapping

strategy, a new mapping strategy or a change to an artefact in the object or the relational silo. Because of the relationship of context between the levels of the framework, the outcome of a dialogue at one level of the framework will also provide the context for a dialogue at another level. Such outcomes, when taken across levels, serve to illuminate the consequences of a solution.

5.9 Summary

There is a mismatch not just because two schemas are described using a different language or abstraction but also because the descriptions of an entity in each schema are not equivalent. Equivalence is important in order to preserve the semantics of an entity in a round-trip between an object-oriented program and a relational database. The technique of equivalence facilitates an understanding of a mapping strategy and informs a dialogue about a mismatch.

An equivalence diagram demonstrates that at the schema level of the framework there is little in common between the semantics of an object and a relational system of identity. Each serves to identify a different concept and so there is a mismatch. The mapping strategy described by Blaha fails to make this distinction and so it is based on a false correspondence. The correspondence implicit in a mapping strategy should be based on how the identity of an entity is described in each schema. An exploration both across the levels and between the silos of the framework identifies the cause of this mismatch.

Keller introduces a surrogate identity in the form of a Synthetic Object Identity. The concept of a Synthetic Object Identity is common to both an object and a relational schema and so could provide an equivalent form of identity. Keller uses a Synthetic Object Identity as the basis for a correspondence of identity in his mapping strategies. However it is possible to interpret a Synthetic Object Identity as a surrogate for an entity or a container. A consistent interpretation of a Synthetic Object Identity both within a silo and between silos is essential for data integrity.

The notion of equivalence presented in this dissertation is not only a schema level concern involving the description of an entity. Reflecting on the contextualisation provided by the levels of the framework, the reference silo is introduced. At each level an artefact in the reference silo provides a point of reference for understanding equivalence between an artefact in the object silo and an artefact in the relational silo. Such an understanding, at a particular level, informs a dialogue to illuminate a mismatch. When taken across the levels of the framework the products of a dialogue inform a dialogue at another level so it is possible to explore the consequences of a solution.

Chapter 6 Identifying the Cause of a Mismatch

6.1 Introduction

The framework provides a structure for exploring the cause of a mismatch. This chapter describes and demonstrates a four-stage process (Ireland, Bowers et al. 2009b) that provides guidance in the use of the framework.

The process is demonstrated using a worked example based on the FTI Case Study in Appendix B. The cause of two mismatches is identified at the concept level and changes are suggested at each level of the framework. The ability to affect a change at a level of the framework depends on the power and influence of those involved.

6.2 The Four-Stage Process

Figure 25 is an outline of the four-stage process that provides context and guidance in the use of the framework to identify the cause of a mismatch. Following the process shifts thinking about a mapping strategy from issues of a correspondence between artefacts in the schema of an object-oriented program and the schema of a relational database to an understanding of the cause of a mismatch, which can be at any level of the framework.

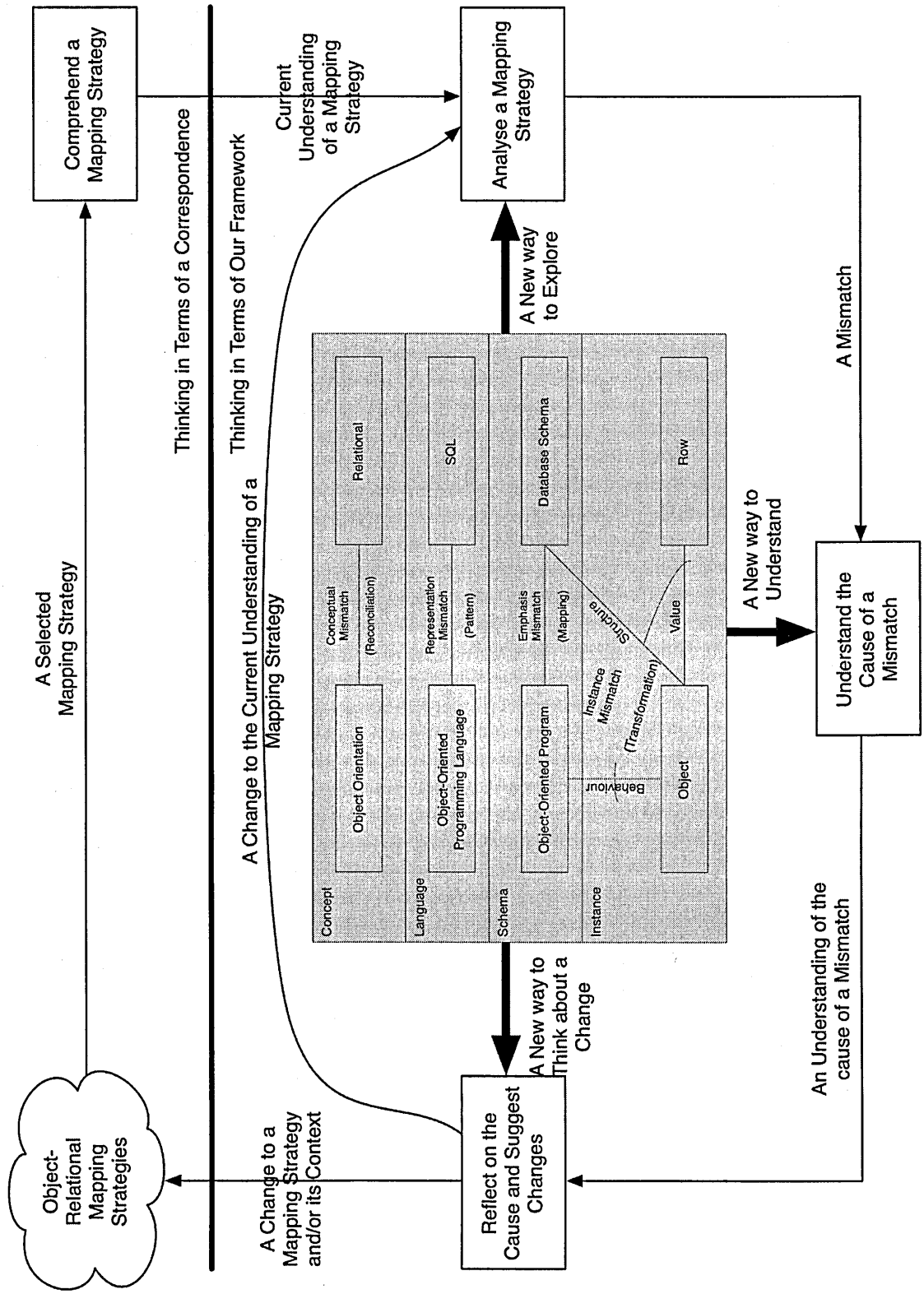


Figure 25 - A Framework Based Approach

The process starts with a choice of a mapping strategy not shown explicitly on the diagram, just implicitly. The process then proceeds in a clockwise direction around Figure 25, through the stages of comprehending a mapping strategy, analysis of that mapping strategy, understanding the cause of a mismatch, and finally suggesting changes to the mapping strategy or the artefacts involved. At each stage of the process the framework is used as a tool for analysis and for the presentation of results.

The next section demonstrates the use of the process to identify the cause of two mismatches. Each stage of the process is illuminated using a worked example. The worked example demonstrates the ways in which the framework is used. The results are options for change in order to address each mismatch.

6.3 Using The Process and Framework

The approach is illustrated using a case study that provides a context for understanding a mapping strategy. A case study can be used to clarify the semantics of a mapping strategy, explain a compromise and illuminate a mismatch but if it is doing it on its own, just by being an example, then there is a risk that it clarifies these things because these things are specific to that case study. Applying a mapping strategy to a case study provides a worked example, demonstrates comprehension and cements an understanding.

6.3.1 Case Study

The case study for the worked example is based on the entity "Instrument" taken from the universe of discourse described in the FTI Case Study in Appendix B. In that case study the entity Instrument is described using a class hierarchy that is reproduced in Figure 26.

Figure 26 shows that there are two distinct and mutually exclusive kinds of instrument: Equity and Debt. Each is identified by an International Securities Identifying Number (ISIN) code. Classes Equity and Debt have no further subclasses. In order to simplify the example, no associations or aggregations have been used.

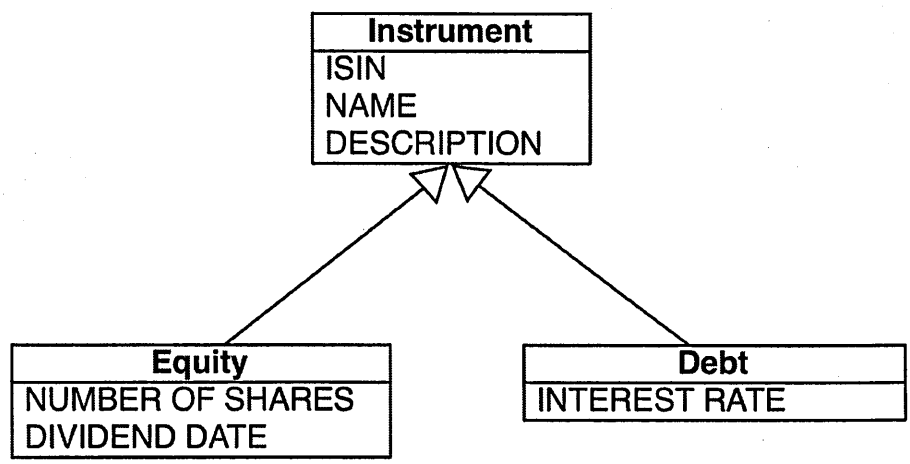


Figure 26 - Financial Instrument Class Hierarchy

Such a hierarchy would form the basis for a Java program that would maintain data about a financial instrument. The detailed design of that Java program is beyond the

scope of this dissertation. For now the assumption will be made that Figure 26 is a suitably accurate and sufficient description of a class model for that Java program.

6.3.2 A Choice of Mapping Strategy

The process starts with the selection of a mapping strategy. The requirement is to provide a means to store data about an object of class Equity and an object of class Debt in a relational database.

Chapter 3 demonstrated that understanding the structure problem is important to understanding object-relational impedance mismatch. A mapping strategy is chosen, introduced in Chapter 2, which represents the structure problem. The mapping strategy embodies a correspondence between a class hierarchy and a representation using SQL:1992.

The literature, for example (Ambler 2003) and (Keller 1997), provides guidance on a choice of a mapping strategy based on costs and benefits. That guidance is used initially but a subsequent choice of mapping strategy will be informed by the outcomes from using the framework.

There are a number of mapping strategies that take as their starting point a class hierarchy and form a correspondence with an SQL:1992 based representation. SQL:1992 is used because no description of this mapping strategy in the literature uses the additional facilities available in later versions of the SQL. One author, Keller

(Keller 1997) p9-17, gives three strategies: (1) one inheritance tree one table; (2) one inheritance path one table; and (3) one class one table. These strategies are described in Table 1 in section 2.5.1.

Let us consider mapping strategy (1) ("the mapping strategy"). This mapping strategy embodies a correspondence between a class hierarchy and a single table in a relational database. A row of this table will store data about an object of a class in this hierarchy. Keller (Keller 1997), p13 recommends this mapping strategy for a small application and Ambler (Ambler 2006a), p240 recommends it for systems with a shallow class hierarchy. The class model Figure 26 meets both of these criteria.

The following sections demonstrate how, in the context of the process, the framework is used to understand the cause of a mismatch and improve the mapping strategy. The outcomes could be used to compare mapping strategies in order to make a choice of mapping strategy. Such a comparison is not shown instead the focus is on understanding the cause of a mismatch and improving the mapping strategy.

6.3.3 Comprehend a Mapping Strategy

The objective here is to understand the correspondence embodied in a mapping strategy. In the first instance such comprehension will be based on the published literature and practical experience. In future the outcomes of a previous analysis of the mapping strategy, or a similar mapping strategy, using the framework will inform an understanding.

```
create table INSTRUMENT(
   ISIN CHARACTER(12) PRIMARY KEY,
   NAME CHARACTER(20),
   DESCRIPTION CHARACTER(40),
   NUMBER_OF_SHARES INTEGER NULL,
   DIVIDEND_DATE DATE NULL,
   INTEREST_RATE FLOAT NULL)
```

Figure 27 - The SQL:1992 Table INSTRUMENT Derived from the Instrument Class Hierarchy

Figure 27 is the description of an SQL:1992 table that represents the data structure of entity Instrument in the schema of a relational database. The table INSTRUMENT will be used to store data about each object of class Equity and class Debt. The column ISIN has been chosen as the primary key.

In order that data about an object of class Equity and Class debt can be stored in a relational database, the mapping strategy embodies a correspondence between the class hierarchy in Figure 26 and the SQL:1992 table in Figure 27. The correspondence is summarised as follows:

- A single table holds data about all objects in the class hierarchy;
- A column in that table corresponds to an attribute of a class in the hierarchy; and
- The data type of a column must correspond to the type of an attribute insofar as it must accept all possible values of that attribute.

A description of the mapping strategy by Keller (Keller 1997) and a description by Ambler (Ambler 2006a) together make the following assumptions:

- It is not necessary to explicitly maintain the parent-child relationship between a class and a subclass in a relational database. This relationship is used only to identify the attributes necessary for the definition of a table;
- An object can be described using a single row of a table;
- The data types of a class attribute and a column are compatible;
- Only that column corresponding to an attribute of a class to which an object belongs is set for a row. All other columns will be set to NULL;
- The correspondence of a class attribute and a column is documented in some form or at least is somehow known by those who must use it; and
- If the data type of a class attribute is changed, regardless of the position of the class in the hierarchy, that change applies to all rows of a table.

Ambler (Ambler 2006a) describes the practical benefits of this mapping strategy:

- Data about all objects is accessible from a single table;
- There is only one table for a programmer to consider;
- The correspondence of a class hierarchy and a single table is a "simple approach"; and
- It is easy to reflect the addition of a new class in the definition of the table should a requirement change.

Ambler and Keller each describe a number of issues with this mapping strategy. One issue is that of classification. In the definition of table INSTRUMENT there must be

some way to differentiate between data for an object of class Equity and data for an object of class Debt. There are at least three options for achieving this:

1. Infer the class of data in a row from the existence of a value in a column (Keller 1997), p13. For example only a row of data for an object of class Equity will have a value for the column NUMBER_OF_SHARES. For a Debt object this column would have a NULL value;
2. Augment the table definition with a new column the value of which indicates the class to which data in a row belongs (Ambler 2006a). For a row representing data about an object of class Equity for example, this column would have the value "EQUITY"; or
3. Use a discriminator value from the universe of discourse in order to differentiate the class of data stored in a row (Keller 1997), p13. Similar to option 1 (but here the actual value, not its presence, is important), and option 2 but a value is stored in an existing column rather than a new one. For example, the value of a Debt ISIN code might include a character indicating that it provides the identity for a debt instrument. This character would indicate that a particular ISIN code identifies data about an object of class Debt.

Let us consider option 1 because it does not require the maintenance of additional data. However in order to use table INSTRUMENT correctly the mechanism for inferring a class must be documented and understood by all involved in the development of an object-relational application. A consequence of option 1 is that it must be possible to differentiate the class of data in a row based on existing

attributes. This strategy is therefore not suitable in the case when a subclass does not introduce an attribute.

Other issues documented in (Keller 1997) and (Ambler 2006a) include potential wasted space in the database through the use of NULL, the consequences of changes to the class hierarchy because all classes are coupled to the same table, locking issues because all data is in the same data space, and indexing issues because secondary indexes are required.

This section has demonstrated that it is possible to achieve some understanding of a mapping strategy based on the literature and on experience. In the next stage of the process the framework is used to analyse the mapping strategy.

6.3.4 Analyse a Mapping Strategy

The objective is to identify a compromise in the correspondence embodied in a mapping strategy and so highlight a mismatch. The concern at this stage is to understand whether a mapping strategy correctly represents a concept and whether any assumptions underpinning the mapping strategy are safe. The result of the analysis is the identification of two mismatches.

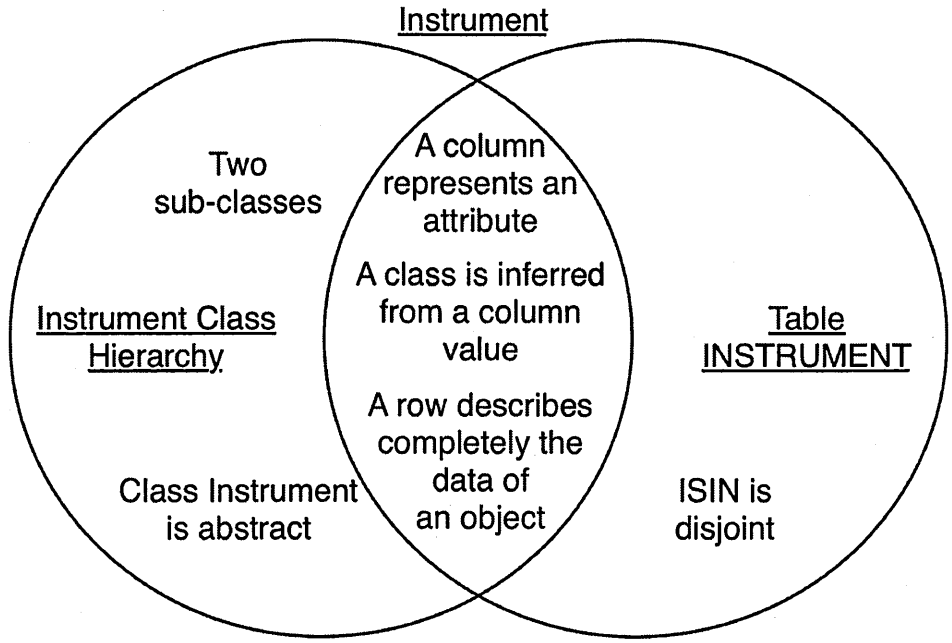


Figure 28 - Equivalent Representations of the Entity Instrument

An equivalence diagram Figure 28 is used to understand the correspondence between the two representations of entity Instrument: the Instrument class hierarchy (Figure 26) and the table INSTRUMENT (Figure 27). This correspondence is embodied in the definition of the mapping strategy. Figure 28 demonstrates that the semantics of the class hierarchy involving the two sub-classes Equity and Debt, the disjoint nature of classes Equity and Debt, and the abstract nature of class Instrument are not preserved by the mapping strategy.

The classes Equity and Debt are disjoint. Class Instrument is abstract. These are design features built into the class model (Figure 26). These semantics are not preserved by the mapping strategy but can be included in the design of a relational database schema. Two examples of a choice of design are described: the first is that of

the semantics of disjoint subclasses Equity and Debt; and the second is the semantics of the abstract class Instrument.

The semantics of a disjoint subclass are preserved indirectly by the choice of primary key. In the definition of table INSTRUMENT the column ISIN was chosen as the primary key. As the primary key ISIN provides the semantics of a disjoint subclass because the value of an ISIN is unique across all financial instruments and there can be only one row in the table INSTRUMENT with a particular ISIN code.

The class Instrument is an abstract class so no object of this class should exist. The semantics of table INSTRUMENT are not the same in this respect and so it is possible to insert a row that represents such an object. That row would have values only for columns ISIN, NAME and DESCRIPTION. Consequently, for reasons of data integrity, it is important to prevent the insertion of a row representing an object of this abstract class in the table INSTRUMENT.

The semantics of the Instrument class hierarchy are not preserved in the table INSTRUMENT. In order to correctly query and maintain data in table INSTRUMENT it is necessary to refer to the class hierarchy (Figure 26). However from the schema ownership problem it is evident that access to the class model and the correct interpretation of that model may be restricted to those involved with the design of an object-oriented program. Two examples are now described; each demonstrates that these semantics are important.

It is not clear from the definition of table INSTRUMENT how to describe a query over parts of the class hierarchy. The semantics of the class hierarchy are used to derive the definition of table INSTRUMENT. However, even in this simple example, without reference to Figure 26 it is not clear from the definition of table INSTRUMENT whether class Equity and Class debt are direct subclasses of class Instrument or whether one is a subclass of the other. Such information is important in order to determine whether a query for data of objects of class Debt should also return data about objects of class Equity.

It is not clear from the definition of table INSTRUMENT to which class a column relates. Each row of the table INSTRUMENT represents data about an object of class Equity or class Debt. In this example the choice was made to infer the class to which data in a row of table INSTRUMENT belongs, but it is necessary to refer to the class hierarchy Figure 26 to identify the columns relevant to a class. In an object-oriented program based on Figure 26 such inference would be unnecessary because its design is based on the class hierarchy.

In combining the definitions of each subclass into a single table this mapping strategy overloads the semantics of a table. According to Date in (Kalman 1994), a relation represents a kind of statement. An SQL table is an implementation of a relation. As a result of the mapping strategy a table must now be interpreted as a more complex kind of statement, because a table now represents more than one kind of statement

one for Equity and one for Debt: although a single row from that table represents a single statement either Equity or Debt.

Based on the above analysis two mismatches emerge:

1. It is not possible to determine those columns that correspond to the class of data in a row without reference to the class model. This mismatch is referred to as overloaded semantics.
2. It is not possible to determine the relationship between data for objects of classes Equity and Debt without reference to the class model. This mismatch is referred to as the semantics of hierarchy.

6.3.5 Understand the Cause of a Mismatch

In this example the objective is to understand the cause of both the mismatch of overloaded semantics and the mismatch of the semantics of hierarchy. The framework is used as both a structure for analysis and for the presentation of the results. The concern at this stage is whether a compromise is necessary because of a mapping strategy or the context in which a mapping strategy operates, whether a compromise is confined to a particular level of the framework, and if not at which level of the framework is the cause of a mismatch.

Table 10 demonstrates that the cause of the mismatch of overloaded semantics is at the concept level of the framework. Starting with the symptoms at the schema level, at the language level Table 10 illuminates why the semantics are not included in the

definition of a table. The table is read from the top down but the analysis proceeds from the schema level up through each level of the framework.

Table 10 - Overloading the Semantics of a Table

Level	Analysis
Concept	A relation represents a single kind of statement.
Language	So SQL requires that all rows in a table share the same definition.
Schema	Consequently using this mapping strategy: There must provide some way to differentiate the class to which data in a row belongs. The columns NUMBER_OF_SHARES, DIVIDEND_DATE and INTEREST_RATE must accept NULL even though for their respective classes they are mandatory. A database constraint must ensure attributes are populated correctly based on a class. A non-key query must include, in the WHERE clause, the criteria for differentiating a row.

All the rows in a table share the same definition, that is, the definition of a table. At the concept level this feature at the language level occurs because a table is based on a relation and a relation represents a single kind of statement.

The framework relates the consequences of a conceptual problem back to the practical problem of wasted space described by Ambler and Keller. The overloading of the semantics of table INSTRUMENT necessitates the use of a NULL valued column. Another consequence is increased complexity of a WHERE clause because it is necessary to differentiate the data in a row.

Section 6.3.3 lists some of the benefits of the mapping strategy. The analysis in Table 10 demonstrates that these benefits come at a cost. Storing data about all objects in a

single table may be a "simple approach" (Ambler 2006a), p240 but it has a cost in terms of work on a database constraint and a query. Whilst Ambler believes it may be easy to add a new class, such a change has consequences including the maintenance of database constraints and queries.

Table 11 - The Semantics of Hierarchy

Level	Analysis
Concept	The semantics of a class hierarchy are well defined, but the actual semantics in use depend on the context provided by a class model and the intention of those who produced it. A relation has no explicit semantics of hierarchy.
Language	So a SQL:1992 table, based on a relation, has no explicit semantics of a hierarchy.
Schema	Consequently using this mapping strategy: The semantics of the Instrument class hierarchy are not present in the table INSTRUMENT. These semantics must be encoded in queries, database views and database constraints.

Table 11 demonstrates that the cause of the mismatch of the semantics of hierarchy is at the concept level of the framework. Again the table is read from the top down but the analysis proceeds from the schema level up through each level of the framework. The semantics of a hierarchy are not present in the table INSTRUMENT because an SQL table does not have explicit support for a hierarchy. An SQL table has no support for a hierarchy because it is based on the concept of a relation which itself has no explicit support for the concept of a hierarchy.

Contrary to the received wisdom, it is not safe to assume that these semantics are unnecessary. In terms of a hierarchy, all that can be said about a single row is that it belongs to a given class and to the hierarchy rooted at class Instrument. Information

regarding the position of that class in the hierarchy is not present in either the data or the definition of the table INSTRUMENT. As a consequence, it is necessary to refer to Figure 26 in order to correctly interpret the data in table INSTRUMENT. These semantics can be encoded in a query, a database view and a constraint. Should that hierarchy change, all places where these semantics are encoded must also be identified and changed.

In summary, thinking in terms of the framework can be useful to identify the cause of a mismatch. Two mismatches have been identified from the correspondence embodied within a mapping strategy at the schema level, and their cause traced to the concept level of the framework. If the focus had been solely on aspects of the mismatch covered by the schema level and instance level of the framework, the real cause of these problems would not have been identified.

6.3.6 Reflect on the Cause and Suggest Changes

Once the cause of a mismatch has been established then improvements to a mapping strategy or to the context in which a mapping strategy operates can be made. In the final stage of the process the framework is used to identify options for change.

Each level of the framework provides an opportunity to address a mismatch in a particular way. The concern at this stage is to understand what change is necessary to address the cause of a mismatch and whether it is appropriate to make a change at another level.

It is possible to improve a mapping strategy in two ways: either directly by addressing the cause of a mismatch or indirectly by addressing the symptoms. A direct intervention will result in an appropriate solution. An indirect intervention, because it does not address the cause of a mismatch, will result in an acceptable solution.

The ability to affect a change will depend on the power and influence of those involved. Those involved with the definition of a standard, such as the SQL standard, or a programming language, such as Java, will have the influence to affect a change at the language level. Others, such as research bodies, might be best placed to deal with a conceptual issue.

If the cause of a mismatch is at the concept level or the language level then those involved in the development of an object-relational application will have to make an indirect intervention. They can address the symptoms of a mismatch at the schema and instance levels because they control the design of an object-relational application. In the case of the two mismatches there are options for both a direct and an indirect intervention.

Four options for an indirect intervention are described and the compromises are highlighted. In the first the use of a different mapping strategy is suggested; in the second and third the possibility of a change to the design of a database schema is

explored; and in the fourth changes to the design of an object-oriented program are considered.

In order to address the issue of overloaded semantics a different mapping strategy can be used. Ambler (Ambler 2006a), p234 describes a mapping strategy that embodies a correspondence between a concrete class and a table. This would remove some of the WHERE clause complexity, wasted space and the need to maintain additional data because the definition of a table corresponds to the definition of a single class, but it would go against both the spirit and the perceived simplicity of the mapping strategy, i.e. to represent all data in a single table. It is the responsibility of those who develop an object-relational application to decide whether such a compromise is acceptable and to understand the consequences of this alternative mapping strategy.

In order to address the mismatch of overloaded semantics a database view can be created for each subclass. A view would embody the criteria for differentiating data about an object of a class and return only those columns relevant to each object of that class. However in order to design such a view those responsible must interpret correctly a class hierarchy, and a view does not always help when new data is to be inserted.

In order to address the mismatch of the semantics of hierarchy a column can be added to table INSTRUMENT to indicate the parent class. However adding such a

column does not solve the problem because it confuses intent and extent. The semantics of a hierarchy are mixed with the data representing an object. This messy implementation fudge is not a viable solution because it is still necessary to know how a hierarchy is structured and there are problems with the representation of an abstract class or any class where an object has yet to be created.

The design of an object-oriented program can be changed to reflect those semantics that can be preserved in a relational database schema. The subclasses Equity and Debt can be removed from the class model. The design of an object-oriented program would then be based on the single class Instrument. Consequently class Instrument could no longer be abstract and the semantics of classes Equity and Debt would need to be included in the definition of class Instrument. Ambler (Ambler 2003), p261 argues that the requirements for an application, rather than a database schema, should drive the design of an object model. Those responsible for the design of an object-relational application must decide whether such a compromise is acceptable.

A direct intervention addresses the cause of a mismatch. Table 12 presents options, in the relational silo, for addressing the cause of the mismatch of overloaded semantics. Table 13 presents options, in the relational silo, for addressing the cause of the mismatch of the omission of hierarchy. Each table demonstrates that addressing the cause of a mismatch provides opportunities for change at other levels of the framework.

Table 12 - Options for a Direct Intervention in the Relational Silo - Overloaded Semantics

Level	Suggestion
Concept	Recognise that a relation can represent more than one kind of statement.
Language	Provide a classifier mechanism in the definition of a table. Extend the SQL standard to support an optional column based on this classifier.
Schema	Provide access to a classifier mechanism within a query.
Instance	Insert only the data values required based on a classifier. Omit a column if it is not relevant to a particular kind of row.

Table 13 - Options for a Direct Intervention in the Relational Silo - The Semantics of Hierarchy

Level	Suggestions
Concept	Recognise the possibility of a hierarchy of relations. Support the concept of an abstract relation.
Language	Support a hierarchy of tables and permit a single query over a hierarchy of tables. That query does not need to include the names of all sub-tables.
Schema	Create a separate table based on classes Instrument, Debt and Equity. Each table is part of a hierarchy of tables. Each table can be queried individually or as part of a hierarchy. Query the entire hierarchy or part thereof using a single statement.
Instance	Create a row in the corresponding base table or by inserting into table INSTRUMENT.

The objective is two fold. First to show that there are options at the concept level; and second to highlight that if the cause of a mismatch is addressed, there would be new options at other levels of the framework whilst maintaining the relationship of context between the levels.

In each case a complete solution is not proposed nor are the consequences of each change at the concept level established. A dialogue would need to establish, at each level, whether a change is acceptable and if so, to understand the consequences for artefacts in both the object and relational silos.

In order that others benefit, and to avoid wasted effort, these suggestions and improvements should be fed back into the wider discourse. The framework provides a structure to communicate these improvements and to facilitate a coherent discourse in spite of the schema ownership problem. At this point a full cycle of the process (Figure 25) is complete.

6.4 Reflection

The process provides guidance in the use of the framework to identify the cause of a mismatch. The framework supports a choice of mapping strategy by asking that those responsible think about the cause of a mismatch and its consequences. As such the process augments any software development cycle at the point where a choice of a mapping strategy must be made.

The worked example demonstrates that a case study is a useful tool that supports both an understanding of a mapping strategy and the illumination of results. In order that an analysis is relevant to a particular object-relational application, it is expected that in practice the case study would be replaced by a real-world example.

The results of the process will inform the next iteration. The worked example involved a number of choices including the case study, the mapping strategy and the mechanism for differentiation. The process can be repeated with a different set of choices in order to understand the consequences of a different case study (or a real-

world example), a different mapping strategy, or a different mechanism for differentiation.

The worked example demonstrates a link between a mismatch of structure and a mismatch of concept first identified by Copeland & Maier. The analysis of this mapping strategy has led to the identification of two mismatches. The cause of both mismatches is at the concept level of the framework. A dialogue at the language, schema and instance levels would be informed by these results.

Because the cause of each mismatch is at the concept level, solutions at the language and schema levels will typically involve a compromise because they must address the symptoms. Consequently, as expected, Amblers mapping strategy involves a compromise. For example, it is necessary to include a way to differentiate data in a row because the strategy overloads the semantics of a table.

Keller (Keller, Jensen et al. 1993) observes that, in practice, a mismatch takes time and effort to address because "developers have to hand code an interface between their objects and their existing relational databases". Keller goes on to describe a code generator to automate the production of an interface. However one reason a mismatch will take time and effort to address is accommodating a compromise, such as a way to differentiate data in a row, and dealing with those semantics that are present in one schema but not in the other.

The framework draws attention to the cause of each mismatch and provides a way to think about a change. Neither Ambler nor Keller refers to the cause of these mismatches in their work on the strategy. However in order to use the framework it is necessary to be able to work across both silos and levels of abstraction. A dialogue supports working across silos but to work across abstractions requires other knowledge within a silo. Such knowledge includes, for example, understanding that an artefact of a language is based on a particular concept level artefact and understanding the consequences, for that language, of a change to that concept level artefact.

The ability to address the cause of a mismatch depends on the power and influence of those involved. If the cause of a mismatch is not at the schema level there are options open to those developing an object-relational application. Whilst these options do not address the cause of each mismatch they can improve the situation in the short term whilst the cause of a mismatch is addressed.

6.5 Summary

Understanding the correspondence embodied in a mapping strategy does help to identify a compromise. Using the framework it is possible to explore the reason for that compromise at different levels of abstraction and identify the cause of a mismatch. Thinking about a compromise at a number of levels of abstraction delivers insights into the cause of a mismatch. These insights provide an opportunity to improve a mapping strategy and the context in which that strategy operates.

The process provides guidance to understanding the cause of a mismatch and to improving a mapping strategy. Options for change were identified that are linked to a conceptual problem not a symptom of an implementation. The process supports a shift in thinking away from implementation issues because it starts by understanding a mapping strategy and issues of implementation, but finishes by suggesting solutions at a number of levels of abstraction.

Effecting a change at each level of the framework involves a different group of people. Changing the definition, or an implementation, of SQL for example is not an option for those developing an object-relational application. However, the definition of SQL is subject to review and to change. In the next chapter the framework is used to understand the consequences, for a mismatch, of the introduction of object-based features in SQL:1999.

Chapter 7 Understanding the Consequences of a Solution

7.1 Introduction

SQL:1999 introduced object-based features into SQL. In this chapter the structure of the framework is used to understand the scope and consequences of this change as a solution to the problems of object-relational impedance mismatch (Ireland, Bowers et al. 2010).

Using SQL:1999 it is now possible to implement three different kinds of database schema each of which introduces a different set of concerns. Care must be taken to avoid introducing a mismatch but guidance in the literature varies at the point where such a decision must be made.

7.2 A Structured User Defined Type

SQL:1999 introduced a number of object-based features into the SQL language (Eisenberg and Melton 1999). These were refined in SQL: 2003 (Eisenberg, Melton et al. 2004). Notably, a Structured Used Defined Type (SUDT) provides the cornerstone of support for a form of object. As the name suggests, an SUDT features in the schema of a relational database.

The definition of an SUDT comprises its name and a collection of attributes and methods. The type of each attribute of an SUDT must be one of the system defined data types, another SUDT or a distinct type. All attributes of an SUDT are private. The

value of an attribute can only be set or retrieved using a method. A method can also be used to refine the semantics of a type. SUDTs can be organised in a hierarchy. A subtype inherits the attributes and methods of its parent type and can introduce new attributes and methods. An SUDT can override a method of its parent type. An SUDT can have only a single parent.

An SUDT can be used either as the type of a column or as the basis for a typed table. A value in a typed column is a structured value with no identity. In contrast a column of a typed table corresponds to an attribute of an underlying SUDT. A row of a typed table represents an instance of the type, and has its own identity. A typed table supports polymorphic queries: a query over a typed table returns instances of that type and any of its subtypes stored in other typed tables defined to correspond to the subclass hierarchy.

The conclusions should not be drawn that because it is now possible to use object-based features in a relational database schema, the problems of object-relational impedance mismatch have been addressed and without consequence. In the next section the structure of the framework is used to understand the consequences of an SUDT.

7.3 The Consequences of a Structured User Defined Type

SQL is a language used to describe a relational database schema, but SQL:1999 introduces object concepts into SQL in the form of an SUDT. SQL:1999 can be thought

of as the product of a pattern dialogue at the language level. That dialogue would have involved those responsible for the SQL standard and was informed by concepts from the object silo.

In this section the structure of the framework is used to understand the consequences, for problems of an object-relational impedance mismatch, of an SUDT. This understanding, when taken across levels, could inform a dialogue at each level of the framework.

7.3.1 A Change to the Relational Silo

At the concept level, Date in (Kalman 1994), argues that equating a class with a relation is conceptually incorrect but in practice, at the language level, this is the basis for the approach that is often used (for example (Keller 1997; Ambler 2006a)).

The SQL standard prior to SQL:1999 did not permit the definition of a new data type beyond the re-labelling of an existing type. As a result, those who develop an object-relational application produced solutions using the only data structure available to them: a table. Date saw this as a flaw in the definition of SQL and not a problem with the relational model. The relational model does not prescribe the domains that can be used.

Date in (Kalman 1994) also argues that the relational model needs no extension in order to support the concept of a class. As long as a correspondence is made between

a class and a domain, using an SUDT as the type of a column, then an object (value) will be treated in the same way as any other domain value. A mapping strategy is trivial because a class is just another kind of domain. Used in this way, an SUDT is simply a change within the relational silo; a change that permits much greater flexibility in the implementation of a data type at the schema level.

7.3.2 A New Way of Addressing A Mismatch

As the type of a column, an SUDT implements at the language level, a correspondence of a class with a domain at the concept level. In this role an SUDT addresses the flaw in SQL highlighted by Date in (Kalman 1994). In other words an SUDT implements, what for Date, is a correct interpretation of a domain. For example, Figure 29 shows a column EQUITY based on a SUDT typeEquity which itself is based on the class EQUITY from the FTI Case Study in Appendix B. A value stored in the column EQUITY will have the attributes ISIN, NAME, DESCRIPTION, NUMBER_OF_SHARES and DIVIDEND_DATE.

```
create table EXAMPLE(
     -- other columns
     EQUITY typeEquity not null)
```

Figure 29 - An SUDT as the type of a Column

As the basis for a typed table, an SUDT represents a class as a table. A row represents data about an object. Table 14 uses the levels of the framework to compare a SQL:1992 table with a typed table. The objective is to understand whether an SUDT,

as the basis for a typed table, provides a new way to address a mismatch. Table 14 also shows that an SUDT has implications at the schema and instance levels of the framework.

Table 14 - A Comparison of an SUDT and a Table-based Representation of a Class

Artefact / Action	Framework Level	Using a table	Using a typed table
Class	Language/ Schema	A table	An SUDT plus the typed table based on it.
Object	Instance	A row of table	Row of typed table
Class Hierarchy	Language/ Schema	A single or multiple tables.	SUDT and typed table hierarchies.
Class Membership	Schema/ Instance	Using some form of class-based tag.	Implicit for a row
New Instance	Instance	Insert a row. A constraint is necessary to enforce membership of a class.	Insert a row. No constraint is necessary.
Retrieve an Instance	Instance	Use a SELECT statement	Use a SELECT statement
Delete an Instance	Instance	Use a DELETE statement	Use a DELETE statement
Identity	Instance	One or more columns provide identity.	Each row has a generated unique ID.
Reference	Schema	A PRIMARY KEY and foreign key constraints.	Use a REF type as a foreign key.
Polymorphism	Instance	A SELECT returns rows from this table. A UNION can be used to combine data from other tables.	SELECT returns rows from this table and any sub-table(s).

Figure 30 shows a typed table EQUITY based on an SUDT typeEquity. A row of the table EQUITY is an object of typeEquity and of no other type because typeEquity has

no subtypes. An SUDT as a typed table therefore provides improved semantics for a correspondence between a class and a table.

```
create table EQUITY of typeEquity(
        ref is ID user generated)
```

Figure 30 - An SUDT as the type of a Typed Table

SQL:1999 introduces other improvements in areas such as data integrity, identity and support for polymorphism, but all of these can be achieved, with some effort and compromise, using an SQL:1992 table based approach. A key difference between a table and a typed table is that the semantics of identity and hierarchy are now implicit in the definition of a typed table. Consequently the correspondence between a class hierarchy and a typed table should involve less compromise than the correspondence between a class hierarchy and a table described in section 6.3.

7.3.3 The Removal of the Relational Silo

An SUDT and a typed table are language structures that allow for the inclusion of object-based features in a relational database schema. In terms of the framework, these structures represent the incorporation of artefacts from the object silo into the relational silo at the language level.

An object model can now be used to describe the schema of both an object-oriented program and a relational database. It is possible that problems, such as those of structure, would then be removed because the design of each schema is based on the

same structure. It is important, therefore, to clarify whether an SQL:1999 schema described using exclusively object features removes the need for a relational silo.

A relational database schema that employs only object concepts can be thought of as an object-oriented schema. Neward (Neward 2007) suggested that one way to avoid some of the problems of an object-relational impedance mismatch is to use an object-oriented database. If that database uses the same programming language as an object-oriented program then in terms of the framework, there would be only one silo: the object silo.

Section 2.5.4 observed that there might be a case for sharing concepts between languages and that this might, somehow, remove a mismatch. SQL:1999 is not however the same language as Java or C++, nor does it have exactly the same semantics. Problems such as those of identity still exist because an object in an object-oriented program is separate from an object in a database. Problems of structure and ownership exist because there is a choice of how an SUDT is used. Hence there is not a single language and SQL:1999 does not remove the relational silo. Furthermore the schema of a relational database can use object features, relational features, or a mixture of both when SQL:1999 is used for implementation.

7.3.4 The Introduction of a New Silo

Using SQL:1999 it is possible to produce one of three kinds of database schema. Each schema is a consequence of a different silo. Each silo brings with it a particular

collection of artefacts with which a correspondence to artefacts in the object silo can be made.

An object is not a relational concept, so it has no place in the relational silo. Consequently SQL:1999 does not remove the relational silo, but it introduces two new silos: the SQL-Object silo and the Object-Relational silo. In the framework each silo can be used in place of the relational silo.

A relational schema contains artefacts based on relational principles. Artefacts in such a schema include a table and a column. An SUDT can also be used but only as the type of a column. A relational schema is part of the relational silo.

An object schema contains artefacts based on object principles. Such a schema would include artefacts such as an SUDT and a typed table but exclude a table. In this kind of schema an SUDT would not be used as the type of a column because a table is a relational concept. A typed table is a new form of table one purpose of which is to support an object model. An object schema is part of a new silo referred to as the SQL-Object silo.

An object-relational schema combines artefacts based on both object and relational principles. Such a schema would use a mix of concepts including object artefacts such as a typed table and relational artefacts such as a table. Contrast this with the single

concept model used in an object or a relational schema. An object-relational schema is part of a new silo referred to as the Object-Relational silo.

7.3.5 Reflection

SQL:1999 is not just a language level change. As the cornerstone of support for object-orientation within SQL:1999, an SUDT addresses some existing concept level issues by reinterpreting the concept of a domain, provides for a new correspondence at the language level, new choices of implementation at the schema level and new ways of interpreting and working with data values at the instance level.

An SUDT is a language level artefact but it has consequences at other levels of the framework. In order to understand, for example, the consequences for a mapping strategy of reinterpreting a domain, a mapping dialogue would engage those involved in the design of an object-relational application. Such a dialogue would focus on the use of an SUDT as the type of a column and the implications for the design of an object-relational application.

Figure 31 uses the framework to illuminate the impact of an SUDT as one aspect of SQL:1999. The shaded rectangle serves to demonstrate that the consequences are not limited to the relational silo but extend to mapping strategies at each level of the framework. Using SQL:1999 there are new choices for the implementation of the schema of a relational database. It is possible to design a database using a relational schema, an object schema or an object-relational schema.

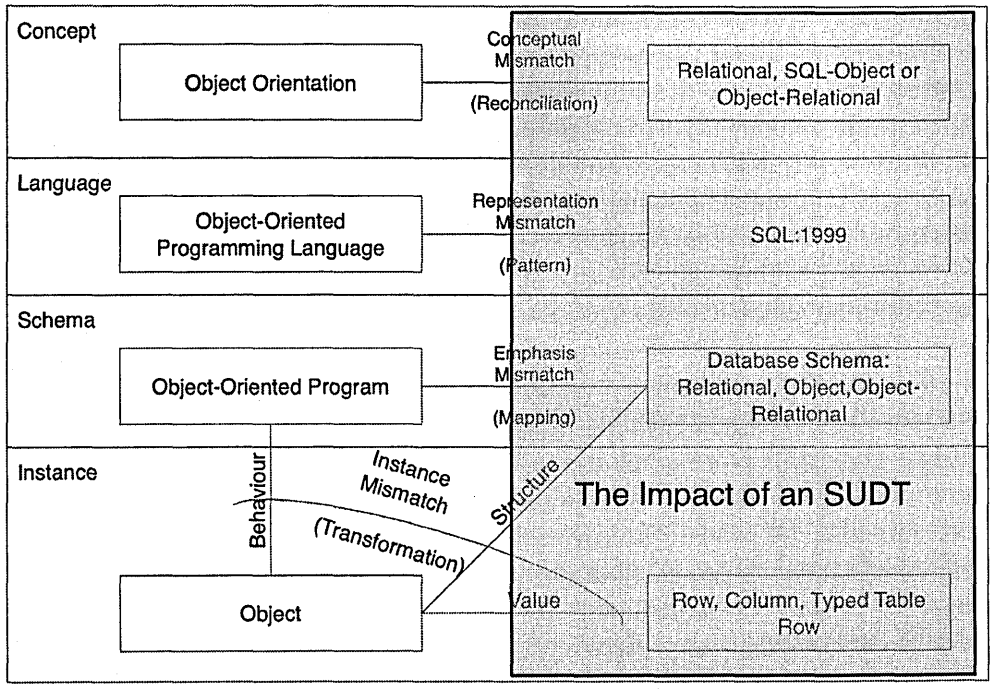


Figure 31 - The Impact of an SUDT

In the design of an object-relational application that uses SQL:1999 a mismatch is expected. SQL:1999 does not remove the relational silo and problems of structure, identity and ownership remain. However, because it is now possible to design three different kinds of database schema using SQL:1999, care must be taken not to introduce a mismatch.

In the next section the framework is used to explore the consequences, for a mismatch, of an object-relational schema. Such a schema will include both relational and object artefacts and so an approach based on an object-relational schema will illuminate issues and choices of implementation that also relate to an object schema. This exploration also demonstrates some of the concerns of a mapping dialogue.

7.4 An Object-Relational Schema

An object-relational schema is an SQL:1999 schema that combines both object and relational artefacts and so mixes concepts. An object-relational database is a database that uses an object-relational schema. The concern here is to understand the consequences, for a mismatch, of choices made during the implementation of an object-relational database.

There is nothing in SQL:1999 to prevent a database schema that mixes concepts. An SUDT can be used as both the type of one or more columns and as the basis for one or more typed tables. Those who implement an object-relational database must understand which to use and when, but then to understand the consequences.

Concepts can be mixed within a single object-relational schema as long as they are equivalent. A class and an SUDT correspond because they represent similar concepts in their respective languages. A correspondence can also be made between a class hierarchy and an SUDT hierarchy. In each case the correspondence does not involve a commitment to the use of a SUDT in the implementation of a database schema. An SUDT can be used as the type of one or more columns and as the basis for one or more typed tables.

Aside from the concern for a mismatch, there are benefits for the design of a database from such an approach. A hierarchy of SUDT provides for reuse, because a subtype

inherits the definition of its parent. This mechanism can be used to implement a consistency of definition within a database schema.

A mix of concepts can lead to the misinterpretation of a schema because two different conceptual frameworks are in use. The value of an object and a statement involving that object are not the same thing. Equating a class with a table implies that a row represents the value of an object, but according to Date in (Kalman 1994), a row of a table is supposed to represent a statement involving an object. The implication is that an object-relational schema should not represent a class with a table but can still use a table for other purposes. An object-relational schema should represent a class with an SUDT.

If concepts are mixed at the schema level, by using an SUDT both as the type of a column and as the basis of a typed table, then problems will follow. One such problem is that of data integrity. There will be problems of integrity because, in terms of the framework, the extent of an SUDT has been split at the instance level: instances of that type now exists both as a value in a column and as a row in a typed table. It is difficult to prevent duplicate data if the extent of an SUDT is split in this way. The problem of integrity is compounded if an SUDT is used as the type of more than one column or as the basis of more than one typed table.

A mismatch of identity will occur if an SUDT is used as the basis for a typed table. Returning to the identity problem explored in Chapter 5, in terms of the framework,

an object is a concept from the object silo and is owned by an object-oriented program: it is created by that program and has an identity in that space. Using an SUDT as the type of a column does not provide for a different space in terms of an identity, because the value of a column does not have its own identity. In this role an object-relational database is complementary to an object-oriented program. Contrast this with the use of an SUDT as the basis for a typed table. A row of a typed table has its own identity and in this respect provides for a different identity space.

There are benefits from using an object-relational schema. However care must be taken when mixing concepts. Employing a single SUDT as both the type of a column and as the basis for a typed table within a single object-relational schema mixes concepts, creates problems of integrity and does not remove a mismatch of identity. In the next section are reviewed three transformations that make use of SQL:1999. The objective is to determine if they provide the guidance necessary to combine concepts and use an SUDT appropriately.

7.5 Transformations Using a Structured User Defined Type

Marcos (Marcos, Vela et al. 2003), Niyomthum (Niyomthum and Chittayasothorn 2003) and Mok (Mok and Paper 2001) each describe a transformation from an object model to a representation using SQL:1999. Whilst each approach the problem in a different way there is a consensus that a class is transformed into an SUDT and a hierarchy of classes is transformed into a hierarchy of SUDT. In this respect each makes an appropriate correspondence of concepts. Where their approaches differ is

on the use of an SUDT; in other words whether to mix concepts. This is the point at which a commitment is made to a particular kind of database schema. It is therefore important that clear guidance is available.

Grant (Grant, Chennamaneni et al. 2006) describes the approach of Marcos (Marcos, Vela et al. 2003) as informal. Marcos requires that each class on a UML class model be tagged in a particular way in order to specify an implementation. It is a choice of transformation whether an SUDT will be used as the type of a column (using the stereotype <<UDT>>) or as a typed table (using the stereotype <<ObjectType>>). It is for those responsible for a transformation to determine whether a class will be tagged as a <<UDT>> or an <<ObjectType>>. The result can be an object or an object-relational schema.

It is left to Grant (Grant, Chennamaneni et al. 2006) to refine the work of Marcos and document that an abstract class is transformed to an SUDT and a concrete class is transformed to a typed table. This means that the typed tables do not properly represent the class hierarchy, so, for example, you cannot query at the highest level. They do not suggest that an SUDT is also used as the type of column and so their preference for an object schema is clear. Contrast this with what Grant (Grant, Chennamaneni et al. 2006) describes as a formal approach adopted by Mok.

Mok (Mok and Paper 2001) uses two algorithms to transform a UML class model into an SQL:1999 schema. The first algorithm removes semantically overloaded elements,

essentially creating a new class for each role. The result is a set of Nested Normal Form (NNF) relations on which they base an SQL:1999 schema. A class introduced through the application of their first algorithm is transformed to a column with the type of the corresponding SUDT. All other classes are transformed to typed tables of the corresponding SUDT. The result is an object-relational schema that mixes concepts but in a consistent way.

Niyomthum (Niyomthum and Chittayasothorn 2003) is concerned with the representation of an existing object-oriented database schema, implemented using InterSystems Cache (InterSystems), as an equivalent SQL:1999 schema. They borrow terminology from their object-oriented database to describe those SUDT that will be used as the type of a column (an embedded class) and those used as the basis for a typed table (a persistent class). They do not question the objective of the design of the object-oriented database schema nor whether that objective is appropriate for an SQL:1999 schema. The result is a schema that can be object or object-relational.

7.6 Reflection

SQL:1999 represents new challenges and choices for the implementation of a relational database schema. The structure of the framework was used to illuminate the scope and consequences of an SUDT as the cornerstone of support for a form of object-orientation within SQL:1999. Also demonstrated were some of the concerns of a mapping dialogue.

SQL:1999 is a hybrid language but it represents a different approach to that adopted in Microsoft LINQ, described in section 2.5.4 . SQL:1999 introduces artefacts that can be used as the basis for a new mapping strategy. Microsoft LINQ (MicrosoftLINQ) achieves its objective, to present data in a relational database as if it were an object, using existing mapping strategies.

An SUDT introduces the possibility of two additional kinds of database schema: one based on object principles; and the other based on a combination of object and relational principles. The term relational database is now overloaded. In the future, when referring to a relational database schema, it is important to make explicit the definition in use.

In order to use an SUDT appropriately guidance is necessary on the use of equivalent concepts. Transformations that make use of SQL:1999 produce an SUDT-based data structure from an object model and make an appropriate correspondence of concepts.

It is important to take care when mixing concepts in an SQL:1999 database schema. However advice on how to use an SUDT varies. Given the same object model, whether the result is an object or an object-relational schema depends on a choice of transformation. In order to avoid a mismatch, that choice must be informed by understanding the consequences of mixing concepts.

A choice of transformation also has consequences for the timeliness of a mapping strategy. Grant (Grant, Chennamaneni et al. 2006) notes that the approach of Mok will produce a different NNF and therefore produce a different SQL:1999 schema depending on the class at which one starts. Consequently a mapping dialogue cannot begin until after a database schema has been produced. Contrast this with the approach of Marcos, where a correspondence is defined on a class model so a mapping dialogue can begin immediately.

7.7 Summary

The framework can be used to explore the consequences of a solution. Because of the relationship of context between the levels of the framework it is possible to understand whether a change at a level provides a basis for a new correspondence at another level. The introduction of an SUDT, at the language level, presents the possibility of three different kinds of database schema. Each schema involves a different mix of concepts and so provides a separate basis for a correspondence with the schema of an object-oriented program.

The introduction of the concept of an object into the design of a relational database schema does not remove all the problems of an object-relational impedance mismatch. Furthermore unless an SUDT is used with care it can introduce problems.

Transformations that make use of an SUDT are consistent in their interpretation of a correspondence between a class and an SUDT. However guidance on the use of an

SUDT varies just at the point in the design of a database schema where particular care should be taken to avoid a mismatch.

Chapter 8 Conclusion

8.1 Introduction

Object-relational impedance mismatch is a complex, costly and time-consuming problem of software development. As new technologies are introduced other mismatches can be anticipated. It is important therefore that there is a way to understand the cause of a mismatch.

This work has looked at problem of software development first identified nearly thirty years ago. Over that period, whilst many are concerned to provide solutions in the form of mapping strategies, only a few have tried to understand the problem. Such understanding is limited to the characterisations of Copeland & Maier and Ambler, and observations of practice by Neward and Ambler. Chapter 3 set out to understand the problem and demonstrated that object-relational impedance mismatch is a complex problem. Chapter 4 detected a latent structure to the characterisations and observations and used that as the foundation for a framework.

Understanding the cause of a mismatch begins by understanding the correspondence embodied in a mapping strategy. Chapter 5 demonstrated a novel technique, based on a notion of equivalence, and used that to analyse two mapping strategies. Such an analysis, when taken across levels of the framework, is used to identify the cause of a mismatch.

The framework provides a structure to understand both the cause of a mismatch and the consequences of a solution. Chapter 6 described a four-stage process that informs others in the use of the framework. By way of a demonstration the process was used to identify the cause of two mismatches. Chapter 7 explored the consequences of SQL:1999 as a solution to the problems of object-relational impedance mismatch and found that whilst SQL:1999 addresses some mismatches, care must be taken not to introduce a further mismatch.

8.2 Summary of Research Products

In summary, the products of this research are as follows:

1. A catalogue of problem themes for making sense of the problem of object-relational impedance mismatch. This work was published in (Ireland, Bowers et al. 2009a).
2. A classification of four kinds of object-relational impedance mismatch. This work was published in (Ireland, Bowers et al. 2009a) and demonstrated in (Ireland, Bowers et al. 2009b).
3. A classification of four kinds of object-relational mapping dialogue. This work was published in (Ireland, Bowers et al. 2009a) and demonstrated in (Ireland, Bowers et al. 2009b).
4. A framework comprising two silos crossed by four levels of abstraction. This work was published in (Ireland, Bowers et al. 2009a), demonstrated in (Ireland, Bowers et al. 2009b) and refined in (Ireland, Bowers et al. 2011).

5. A technique based on a notion of equivalence. An equivalence diagram is used to explore the correspondence embodied in a mapping strategy in order to identify the reason for a compromise and so illuminate a mismatch. This work was published in (Ireland, Bowers et al. 2011).
6. A four-stage process that, when combined with the framework, shifts thinking about a mapping strategy from issues of a correspondence between two schemas to an exploration of the cause of a mismatch at other levels of abstraction. This work was published in (Ireland, Bowers et al. 2009b).
7. A classification of three kinds of relational database schema using SQL:1999 a version of SQL that introduced object-based concepts into the relational database language. This work was published in (Ireland, Bowers et al. 2010).

8.3 Research Question – The Answer

Section 1.3 presented the research question and the hypothesis tested in this dissertation. The research question was: can a more encompassing perspective on object-relational impedance mismatch be developed, one that provides actionable insights into the cause of a mismatch? Exploring the hypothesis led to three sub-questions. This section examines the extent to which the research question and the sub-questions have been answered and the hypothesis has been tested.

8.3.1 Complete and Consistent Questions

The research hypothesis was that a general framework based on a synthesis of both the theory and practice of impedance mismatch provides a foundation for

understanding the cause of a mismatch. The following are prerequisites in order to test this hypothesis:

1. Expose the different perspectives of impedance mismatch;
2. Identify the underlying concern of each perspective;
3. Relate those concerns in a homogenous structure that supports an investigation into the cause of a mismatch.

These prerequisites provide the rationale for the first sub-question: in what way does the framework reconcile the different perspectives in the literature?

8.3.2 Reconciling Different Perspectives

Chapter 2 exposed the different perspectives of impedance mismatch in the literature. Chapter 4 demonstrated the concern of each perspective and synthesised those concerns in the form of a four-level framework. The framework reconciled these perspectives and demonstrated that each concern represents a separate level of abstraction over an object-relational application.

Each level of the framework provides the context for a choice of implementation made in the level below. A mapping strategy is a choice of implementation that typically involves a compromise. Exposing that compromise by exploring the context of a mapping strategy across the levels of the framework provides the rationale for understanding the cause of a mismatch.

Chapter 3 highlighted that a focus on problems and solutions alone, typical of approaches to date, does not help to identify the cause of a mismatch. Consequently it is necessary to take a different approach, one that shifts the emphasis from a concern with how to make data in an object fit into a relational database to understanding the reason why data in an object does not fit seamlessly into a relational database. The answer to this question provides the motivation for the second sub-question: can the framework be used to drive an analysis process, and if so, how might that process be structured?

8.3.3 An Analysis Process

An object-relational mismatch occurs when an object-oriented program uses a relational database for storage. There is no mismatch between an object-oriented program and a relational database until a decision is made to use a particular mapping strategy. At that point a commitment is made to store the data of an object in a relational database and in a particular way. Typically a compromise is necessary and the transfer of data between an object-oriented program and a relational database is not seamless.

Chapter 6 described and demonstrated a four-stage analysis process that uses the framework and the technique of equivalence as tools to explore the consequences of a choice of mapping strategy. During the analysis stage an equivalence diagram exposes a compromise and in the next stage the levels of the framework are used to expose the reason for that compromise and consequently the cause of a mismatch.

Using the framework it is now possible to identify the cause of a mismatch as long as a difference of semantics, a difference of implementation, a difference of language or a difference of conceptual framework is the cause. Chapter 6 demonstrated that the cause of a mismatch of structure is at the concept level. Not all mismatches are caused at the concept level and Chapter 5 demonstrated that the cause of a mismatch of identity is at the language level. Consequently the framework relates the symptoms of a mismatch to a difference of concept or a difference of language first acknowledged by Copeland & Maier (Copeland and Maier 1984).

During the final stage of the process the framework provides a structure for a new way to think about the scope and consequences of a solution. Chapter 6 concluded that the ability to affect a solution depends on the power and influence of those involved. Identifying options for change provides the motivation for the third sub-question: Does considering the cause of a mismatch provide actionable insights into a solution?

8.3.4 Actionable Insights into a Solution

If the cause of a mismatch is not known then it is not possible to be certain that a solution is appropriate or whether it is somehow acceptable. If the cause of a mismatch is not at the schema level then a programmer must make an acceptable solution i.e. one that involves a compromise. Chapter 5 used the framework both to locate the cause of a mismatch of identity at the language level and to demonstrate the importance of a consistent interpretation of a concept across silos. Chapter 6

further demonstrated that two mismatches of structure are caused at the concept level. In both examples a compromise is necessary in a solution at the schema level because, as the framework highlights, the cause of each mismatch is not at the schema level.

Chapter 7 explored the implication of object-based features in SQL:1999. It is conceivable that the introduction of object-based features into SQL might somehow remove a mismatch of structure and identity because the concept of an object can be used in the design of both a program and a database. The levels of the framework were used to explore the consequences of an SUDT as one solution to the problems of structure and identity identified in Chapters 5 and 6.

Thinking in terms of the framework does provide actionable insights. For example, Chapter 7 concluded that care must be taken when using an SUDT to address problems of hierarchy and identity as further problems of identity and problems of structure and ownership remain. In the case of the problem of identity problems remain because an object and a row of a typed table are different concepts. As another example, Chapter 5 demonstrated the importance of a consistent interpretation of a concept across silos.

8.3.5 Testing the Hypothesis

In order to test the hypothesis it must be established that:

1. The framework does provide a way to identify the cause of a mismatch; and

2. The characterisations of Copeland & Maier, Neward and Ambler cannot be used individually to identify the cause of a mismatch.

To identify the cause of a mismatch it is necessary, but not sufficient, to understand that Copeland & Maier, Ambler and Neward each characterise the same problem. Essential to understanding the cause of a mismatch is the recognition that each author describes the same problem but as a different abstraction of an application.

Recognising such an organisation of perspectives prompts questions about cause and effect across the characterisations that have not been asked in the literature. For example is the "object-to-table mapping problem" described by Neward caused by a mismatch of language or a mismatch of concept described by Copeland & Maier? The framework provides a structure that highlights such a question and provides a way to explore the answer.

The four levels of the framework unify the concerns of Copeland & Maier, Ambler and Neward. Each level of the framework represents a different concern. These concerns are now recognised as not separate but related and in a particular way. This dissertation has demonstrated that the framework organises these concerns in a useful way. The organisation of both theory and practice in this way has not been attempted in the literature. Furthermore, systematically working through the concerns of Copeland & Maier, Ambler and Neward individually would not have identified the cause of a mismatch; Chapter 3 demonstrated that relationships

between problems, such as those described by Neward, does not lead to the cause of a mismatch.

8.4 Benefits of Applying the Framework

The framework shifts thinking about object-relational impedance mismatch from a concern with a solution to a concern with understanding the cause of a problem. Only once the cause of a problem has been identified can it be known whether a solution is appropriate or somehow acceptable. Chapters 6 and 7 demonstrated that using the framework it is now possible to work toward an appropriate solution in a structured way. Starting with the compromise embodied by a mapping strategy the framework is used to explore why that compromise is necessary and the consequences of a solution before it is implemented.

The framework extends the scope for a solution from a choice of implementation made by a programmer to other levels of abstraction and to other stakeholders. A solution to a mismatch can involve a change at more than one level of the framework and across silos. Such a solution will involve a number of stakeholders such as those responsible for a programming language and those responsible for the design of a database. The framework provides a structure to coordinate their activities; a structure to assign responsibilities; a technique for making sense of the issues; and a language for communicating accurately and consistently about an issue and the possibilities, for a solution, of both a direct and an indirect intervention.

An implication of the framework is that it is necessary to develop skills both across levels of abstraction and between silos. This has consequences for the education of those responsible for the development of an object-relational application and for those responsible for the technologies used. Stakeholders must be able to communicate effectively both across silos about object and relational technologies and across levels of abstraction. For example, in one conversation a stakeholder could discuss the consequences for a mismatch of a choice of design made in the development of an object-oriented program, whilst in another conversation the same stakeholder could discuss the implications of a change to SQL and the consequences of that change both for the design of a relational database and a mismatch.

8.5 The Applicability of the Framework

Object-relational impedance mismatch is the term used to refer to a difference between the schema of an object-oriented program and the schema of a relational database. In this dissertation it has been demonstrated that the framework provides insights into this particular problem. However the definition of impedance mismatch used in this dissertation is one particular interpretation of impedance mismatch.

It is possible to envisage other kinds of mismatch between other kinds and forms of schema and in so doing adopt a broad interpretation of impedance mismatch. Software integration is the activity of linking together software systems so they act as a coordinated whole. Problems of software integration exist for example at the junction of two sub-systems such as a program and a database, at the junction of two

applications such as order processing and accounting, and at the junction of two programs such as order entry and order processing. Consequently addressing problems of software integration is an important activity in the development of a computer software system.

Impedance mismatch can be viewed as a particular sub-problem of software integration. In this dissertation the problem of integrating two sub-systems, namely an object-oriented program and a relational database, are explored. The question remains to what extent the framework is applicable to other examples of software integration such as between two applications or between two programs.

Whether the framework is applicable in each situation depends to an extent on the choice of implementation language. Two applications may be written using the same programming language and so in terms of the framework the only difference is one of design at the schema level. In this scenario the concept and language levels of the framework are of limited use because at these levels there is no separation of concerns. However the technique of equivalence can be used to explore differences between the two schemas for the purpose of data exchange.

Even if two programs employ the same programming language and the same schema they may still use a mechanism for data exchange that introduces a mismatch. For example one program may expect a single record whilst the other would like to send many records or one program may send data in a format that must also be

interpreted correctly by a receiving program. The schema level of the framework can be used to explore the consequences for a mismatch of each such choice of implementation. Furthermore if the data structure used for data exchange is defined using another language, then all the levels of the framework can be used to explore a mismatch between the schema of that program in one silo, and a data structure used for data exchange in the other silo.

8.6 Research Implications

Chapter 3 demonstrated that the problems of object-relational impedance mismatch are not independent and so work towards a solution must proceed in an integrated way. Chapter 5 demonstrated that an identity problem is caused by a difference between artefacts in an object-oriented program and a relational database. That difference is characteristic of the structure problem. Chapter 7 demonstrated that a hybrid language such as SQL:1999 can address a difference of data structure. As two languages converge in this way an identity problem may then be resolved. However unless care is taken in the design of an object-relational database schema a mismatch of identity can be introduced.

Addressing a mismatch can now proceed in a structured and consistent way, not just across levels of abstraction but between silos. The framework will help to bridge the cultural impedance mismatch described by Ambler (Ambler 2006b). Chapter 2 established that each role and each team involved in the development of an object-relational application has a different priority. Through the use of common levels of

abstraction the framework facilitates a dialogue between proponents of object and relational technologies. Problem themes and the classification of object-relational impedance mismatch provide a language for such a dialogue. A specific set of terms must be employed in a dialogue at each level of the framework and Chapter 4 provided some initial guidelines.

There are latent issues where definitions of a mapping strategy cross abstractions. Using the framework and the language of the classification it is possible to identify such issues. Chapter 4 demonstrated the benefits of a consistent use of terms within each level of the framework. Ambler does not make clear his definition of a class so his correspondence of a class and a table is ambiguous. Such a consistent use of terms will improve the fidelity and the integrity of future work. Fidelity can be achieved because the basis for a correspondence is made clear. Integrity is improved because it is necessary to think about a specific artefact at a particular level of abstraction.

It is not safe to assume that all people interpret a concept in the same way. Different interpretations of a concept, such as an object, are possible both at each level of the framework and between silos. Chapter 6 demonstrated that because it is possible to interpret a concept, such as a table, in different ways there is choice of a mapping strategy. Chapter 5 demonstrated that different interpretations of an object are possible at each level of the framework, and that problems occur if the same interpretation of a concept is not used across silos.

It may not be necessary to transform all artefacts on an object model to a representation in a relational database schema. A hierarchy of classes is part of the schema of an object-oriented program and provides a number of benefits. Unless the same benefits are required in the design of a relational database the transformation of a class hierarchy may be unnecessary. Chapter 6 demonstrated that the correspondence of a class hierarchy and a representation using a single SQL:1992 table creates a mismatch. Chapter 7 demonstrated that the introduction of a form of hierarchy in SQL:1999 introduces other problems.

There are other causes of a mismatch than those described by Copeland & Maier. A choice of design or different interpretations of a concept can also produce a mismatch. Consequently it is not safe to assume that using the same language for a program and a database will remove a mismatch nor will the incorporation of a concept from one language into another in the way suggested by Schwartz (Schwartz and Desmond 2007) and Meijer (Meijer 2006). For example it is not certain that a hybrid language will remove a mismatch. Chapter 7 explored the introduction of an object-based feature in a relational database schema and found that problems of identity and integrity remain both within a database and between a database and a program.

It is possible to improve both a mapping strategy and the guidance for a choice of mapping strategy. An equivalence diagram is used to explore the correspondence embodied in a mapping strategy and so understand a compromise. Chapter 6

demonstrated that the process provides the necessary guidance to improve a strategy at each level of the framework. Options for change were described at each level of the framework. An option for change was linked to a conceptual problem not a symptom of an implementation. The process supports a shift in thinking away from issues of implementation because it starts by understanding a strategy and issues of an implementation, but finishes by suggesting solutions at a number of levels of abstraction. Chapter 6 concluded that because the semantics of a hierarchy are not present in a table there are consequences for the complexity of both a query and a constraint. These consequences inform a choice of mapping strategy.

Contrary to the suggestion of Fussell (Fussell 1997) the decoupling of a program and a database is not a solution to a mismatch. Problems occur when the two are combined and they do not go away using a persistence layer such as Hibernate (Hibernate) or TopLink (TopLink) or a hybrid language such as Microsoft LINQ (Schwartz and Desmond 2007). A persistence layer embodies a number of mapping strategies. Such a layer will only address the cause of a mismatch if the mapping strategy employed is an appropriate solution.

A change to a language is not made in isolation. Such a change can introduce a new and unintended mismatch. The framework provides a structure to understand the consequences of a solution to a mismatch. Chapter 7 demonstrated a consequence of introducing object-based features in SQL:1999. It is now possible to mix concepts in a relational database schema and the result can be one of three different kinds of

database schema. Guidance on the use of mixed concepts varies but such guidance is important because the use of mixed concepts can introduce a mismatch.

In order to resolve a mismatch it can be necessary to address issues of correspondence using separate dialogues across the levels of the framework. This is not the same as a single solution that crosses levels of abstraction or one that considers object and relational issues in isolation such as (Fussell 1997). Such a solution lacks the clarity provided by the framework. A solution to a mismatch can have a consequence within other levels of the framework. However a dialogue must only deal with issues at the corresponding level of abstraction but will be informed by the outcomes of a dialogue at another level.

The dimensions of the framework can be used to identify a gap in the discourse. A level of the framework provides a context for the levels below. In order to address an emphasis mismatch a programmer must translate a correspondence at the language level to a correspondence at the schema level. The framework highlights that in order to do this a programmer must work across abstractions. Chapter 7 demonstrated that a change at the language level introduces new options for the design of a database schema. In the literature the concern of a mapping strategy is with a correspondence of artefacts at the same level of abstraction and not issues across levels of abstraction.

Section 5.7.3 established that a parallel could be drawn between the schema level of the framework and level M1 of the MDA. However the objective of the MDA and the

framework are orthogonal but complimentary. The MDA is concerned with the definition of a transformation that leads to a schema. A transformation defined at level M2 of the MDA can be applied to a model at level M1 in order to produce a new model at level M1. For example, such a transformation might produce the schema for an object-oriented program from a UML class model, whilst another transformation might produce the schema for a relational database from the same UML class model or from a conceptual data model.

The framework is concerned with differences between two schemas, each of which is the product of a transformation. Each transformation may be defined within the MDA although such a definition is not necessary in order to use the framework. In the context of the framework, an equivalence diagram at the schema level could be used to explore the consequences, for a mismatch, of a transformation defined within the MDA. Conceivably suggestions could then be made for a change to the artefacts that are produced by a transformation and by implication a change to the definition of a transformation within the MDA. However such an analysis is not demonstrated in this dissertation.

The framework is concerned with artefacts of implementation, specifically those artefacts used to construct an object-relational application. Consequently it is of limited use when the cause of a mismatch is a choice of transformation made earlier in a process of software development such as a choice of transformation made in the design of a conceptual data model. This is because transformations, such as those

described in Chapter 2, are orthogonal to the framework at the schema level and so it is only possible to explore the consequences of a transformation or a sequence of transformations as they materialise in a schema.

If a mismatch is caused by a choice of transformation, made during a sequence of transformations, then the framework cannot be used to identify where in that sequence the mismatch is introduced. Furthermore the framework cannot be used to understand why a transformation was chosen and whether that choice of transformation is the reason for a mismatch.

The framework can only be used to identify the cause of a mismatch once a choice of mapping strategy has been made. The symptoms of a mismatch will become apparent at the schema level because this is the level at which a programmer must implement a mapping strategy. However mismatches are also introduced at the schema level. Chapter 7 highlighted that a transformation can produce a different database schema from the same UML class model depending on where in that model the transformation starts. Grant (Grant, Chennamaneni et al. 2006) identifies that a choice of a transformation and not just a choice of an abstraction will affect the form of a schema. Consequently a choice of transformation impacts the timeliness of a mapping strategy. The earlier a mapping strategy can be defined in the development of an object-relational application, the sooner all concerned can engage in a dialogue and understand a mapping strategy and its consequences for a mismatch.

The framework is not software architecture nor is it a new mapping strategy. Because the framework is not a solution to the problem of object-relational impedance mismatch there may be reluctance, amongst those responsible for delivering an object-relational application, to embrace the ideas and techniques. However it is a false economy to spend time implementing a solution without first understanding the cause of a mismatch. Consequently education in the benefits of the framework will play an important role in its acceptance as a new perspective on impedance mismatch.

The framework is as much about people as it is technology. In order to use the framework it is necessary to understand both the artefacts at each level and the artefacts in each silo. Such knowledge may be difficult for an individual to acquire and they may not have the power necessary to implement a change. Consequently a dialogue will involve a number of stakeholders and consensus both on the exact nature of a mismatch and the cause of that mismatch may be difficult to achieve because those responsible may be reluctant to share ideas and to make changes.

8.7 Concluding Remarks

The research products and the contribution of this dissertation provide a new way to think about a mismatch and how to go about addressing it. A mismatch should not be accepted as a fact of life as suggested by Keller (Keller 1997) without first understanding the cause. What appears as a chasm (Brown and Whitenack 1994) or a quagmire (Neward 2006) may be a consequence of understanding. A mismatch may

or may not be inevitable. Once the cause of a mismatch is understood only then will it be clear what must be changed for an appropriate solution, whether it can be changed and the nature of that change.

Chapter 9 Future Work

9.1 Introduction

The research products embody a new perspective on object-relational impedance mismatch. In this dissertation the products are developed and their usefulness is demonstrated using examples based on the important problem themes of schema ownership, structure and identity. This chapter explores the opportunities for future work in terms both of the framework and the possibilities it presents.

9.2 Opportunities

The framework presents a number of opportunities for future work. The following sections describe each opportunity and consider how both the research products and their outcomes can be extended to frame research in other areas both in computing and outside the discipline.

9.2.1 The Problems of Object-Relational Impedance Mismatch

Chapter 3 described a number of different problems of object-relational impedance mismatch. In order to demonstrate the validity of the framework it was sufficient to demonstrate that the framework can produce new insights into some of these problems. It was not necessary to explore or to identify the cause of every mismatch.

In order to better understand object-relational impedance mismatch the research products should be applied to other problems. In addition to problems of structure,

schema ownership and identity not covered in this dissertation, such problems include those of processing model, encapsulation and instance described in Chapter 3.

9.2.2 A Process of Software Development

The framework is concerned with artefacts of implementation. It does not address specifically the process that produces an object-relational application. However a choice made in a transformation, as part of such a process, has a consequence for a schema. It may then also have a consequence for a mismatch.

Chapter 2 describes a two-stage process of software development. The first stage, the conceptual stage, results in a conceptual model of a universe of discourse. The second stage, the implementation stage, results in an implementation model based on a conceptual model. Between stages a choice of transformation is made that will ultimately influence the schema of an object-oriented program and the schema of a relational database.

In order to explore and to understand the consequences of a transformation for a mismatch a third dimension could be introduced to the framework. Recognising the separation of silos at four levels of abstraction, a third dimension would reflect each stage of a process of software development such as that described in Chapter 2. For example, at the conceptual stage such a framework would recognise, in separate silos, a conceptual object model and a conceptual data model at the schema level. At the language level would be the language used to describe each conceptual model. It

would then be possible to explore the cause of a mismatch between a conceptual object model and a conceptual data model.

Whilst a third dimension to the framework is purely speculative it would reflect distinct stages of model development such as conceptual and implementation. A third dimension could also be used to explore and to understand the consequences, for a mismatch, of a particular transformation or sequence of transformations between models at each stage of software development. For example in a third dimension it would be possible to explore the consequences, for a relational database schema, of a choice of abstraction in a conceptual data model and a choice of transformation to an implementation model. Further work is necessary to establish the nature and possibilities of a third dimension.

9.2.3 The Description of an Entity

Chapter 5 explored ways in which an entity from a universe of discourse might be described. However in terms of equivalence it is not sufficient only to describe an entity; that description must not favour one conceptual framework over another. This is because a description of an entity that favours one conceptual framework can introduce a mismatch between that description and the description of the entity using the other conceptual framework. Such a mismatch would distract from the objective of equivalence, that is, to understand the correspondence embodied in a mapping strategy and any compromise that is necessary to enable that correspondence.

Chapter 5 noted that the challenge is that the language for the description of an entity must be a generalisation of an object and a relational language. Further work is necessary to understand both how the language of a generic conceptual model, the products of Multi-paradigm Modelling and the MDA can be used to describe an entity in such a way, and how such a description can then be used to explore equivalence.

9.2.4 A Formal Ontology of Terms

The framework helps to address the cultural impedance mismatch described by Ambler (Ambler 2006b). Through the use of a dialogue at common levels of abstraction, the framework facilitates a discourse between proponents of object and relational technologies.

However for reasons of consistency, of precision and of clarity a specific set of terms must be employed at each level of the framework. Chapter 4 presented some initial guidelines but further work is required to develop a formal ontology of terms both within a silo and between silos at each level of abstraction.

9.2.5 A Hybrid Language

Chapter 6 explored a consequence of the introduction of object-based features into SQL. It is clear that SQL:1999 provides new opportunities for addressing a mismatch. However care must be taken when mixing concepts.

The framework can be used to explore the consequences of a hybrid language but it is not clear at what point a language ceases to be object or relational. The question

remains whether sharing other concepts in a hybrid language can provide new opportunities for addressing a mismatch.

9.2.6 Exploring Another Impedance Mismatch in Computing

An object-relational impedance mismatch is one form of impedance mismatch. In this dissertation the two silos in the framework were used to understand issues of correspondence between object and relational artefacts.

One of the silos could be replaced with another silo in order to understand other forms of impedance mismatch. For example a mismatch could be explored between an object-oriented programming language and a technology such as NoSQL (Couchbase 2011). If both silos are replaced a mismatch between a procedural language and an object-oriented database could be explored.

A generalised form of the framework could help to understand issues at the junction of any two conceptual frameworks used in software development. In such a generalised framework an artefact at each level of abstraction would describe a silo based on a particular conceptual framework. The process in conjunction with the framework and the notion of equivalence presented in this dissertation could be used to understand a correspondence between two such artefacts.

9.2.7 Exploring Impedance Mismatch Outside Computing

A generalised version of the framework might be used to explore differences between conceptual frameworks outside computing. There are many languages and many different models in disciplines outside computing.

Consider for example a spoken language. The framework might be used to understand a difference between a description of some thing in Japanese and a description of the same thing in English. It would then be possible to explore, for example, whether a problem of communication is down to a difference of culture, a difference in the semantics of a language or a difference in the semantics of a particular description at the schema level.

9.3 Closing Remark

The problem of object-relational impedance mismatch is important in practice because a mismatch costs in time and effort to address. An acceptable solution will give the illusion that a mismatch is solved. Such a solution might then reinforce the belief that something is being done to improve the situation even though the cause of a mismatch has not been addressed. However the conclusion should not be made that a mismatch is inevitable and that there is no alternative but to deal only with the symptoms.

This dissertation provides a new way to understand object-relational impedance mismatch based on a synthesis of both theory and practice. The research products

provide a new way to address the problems of object-relational impedance mismatch. It is now possible to take a step back from issues of implementation in order to understand object-relational impedance mismatch and to explore the cause of a mismatch.

The possibilities of a framework-based approach to an understanding of object-relational impedance mismatch are many and varied. The research products are a foundation from which to pursue future research. It is possible to explore other problems and the cause of other mismatches; develop a formal ontology of terms that will aid both precision and communication; and seek to focus on that which is important: the preservation of the semantics of an entity.

References

Amaral, V., C. Hardebolle, et al. (2010). Recent Advances in Multi-paradigm Modeling. Models in Software Engineering. Berlin, Springer-Verlag. **6002/2010:** 220-224.

Ambler, S. (2005, June 21st 2005). "The Design of a Robust Persistence Layer for Relational Databases." Retrieved 10th May 2007, from http://www.ambysoft.com/downloads/persistenceLayer.pdf.

Ambler, S. (2006a, 6th October 2006). "Mapping Objects to Relational Databases: O/R Mapping In Detail." Retrieved 11th January 2010, from http://www.agiledata.org/essays/mappingObjects.html.

Ambler, S. (2006b). "The Cultural Impedance Mismatch Between Data Professionals and Application Developers." Retrieved 10th May 2007, from http://www.agiledata.org/essays/culturalImpedanceMismatch.html.

Ambler, S. W. (2003). Agile Database Techniques - Effective Strategies for the Agile Software Developer, Wiley.

An, Y., X. Hu, et al. (2010). "Maintaining Mappings between Conceptual Models and Relational Schemas." Journal of Database Management **21**(3): p36-68.

Armstrong, D. J. (2006). "The Quarks of Object-Oriented Development." Communications of the ACM **49**(2): 123-128.

Bauer, C. and G. King (2007). Java Persistence with Hibernate, Mannig Publications Co.

Bezivin (2001). From Object Composition to Model Transformation with the MDA. 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39). Santa Barbara, California, IEEE Computer Society: p350.

Blaha, M. R., W. J. Premerlani, et al. (1988). "Relational database design using an object-oriented methodology." Communications of the ACM **31**(4): 414-427.

Bowers, D. (2003). "Detection of Redundant Arcs in Entity Relationship Conceptual Models." Lecture Notes in Computer Science **2784**: p275-287.

Brown, K. and B. G. Whitenack. (1994). "Crossing Chasms: A Pattern Language for Object-RDBMS Integration "The Static Patterns"." Retrieved 30 December 2008, from http://www.ksc.com/articles/staticpatterns.htm.

Brown, P. (2001). Object-Relational Database Development: A Plumber's Guide, Informix Press.

Cabibbo, L. and A. Carosi (2005). "Managing Inheritance Hierarchies in Object/Relational Mapping Tools." Lecture Notes in Computer Science **3520**: 135-150.

Cabibbo, L. and R. Porcelli (2003). M2ORM2: A Model for the Transparent Management of Relationally Persistent Objects. Database Programming Languages: 9th International Workshop, Potsdam, Germany, Springer Berlin / Heidelberg.

Capretz, L. F. (2003). "A Brief History of the Object-Oriented Approach." Software Engineering Notes **28**(2): 1-9.

Chen, P. P.-S. (1976). "The Entity-Relationship Model - Toward a Unified View of Data." ACM Transactions on Database Systems **1**(1): 9-36.

Childs, D. L. (1968). Feasibility of a set-theoretical data structure - a general structure based on a reconstituted definition of relation. IFIP Cong., Amsterdam, North Holland Pub. Co.

COBOL. Retrieved 14th December 2011, from http://www.cobol.com

Codd, E. F. (1970). "A relational model of data for large shared data banks." Communications of the ACM **13**(6): 377-387.

Cook, W. R. and A. H. Ibrahmi. (2005, March 2011). "Integrating Programming Languages & Databases: What's the Problem?", from http://www.odbms.org/download/010.03%20Cook%20Integrating%20Programming%20Languages%20and%20Databases_What%20is%20the%20Problem%20September%202006.PDF.

Copeland, G. and D. Maier (1984). "Making Smalltalk a database system." ACM SIGMOD Record **14**(2): 316-325.

CORBA. Retrieved 5 September 2011, from http://www.corba.org/.

Couchbase. (2011, May 2011). "NoSQL Database Technology." from http://www.odbms.org/download/NoSQL-Whitepaper-1.pdf.

Date, C. J. (1985). An Introduction to Database Systems - Volume 2, Addison Wesley.

Date, C. J. (1986). An Introduction to Database Systems - Volume 1, Addison Wesley.

Davies, I., P. Green, et al. (2004). Conceptual Modelling – What and Why in Current Practice. Proceedings Conceptual Modeling – ER 2004: 23rd International Conference on Conceptual Modeling, Shanghai, China.

Dieste, O., M. Genero, et al. (2003). "A conceptual model completely independent of the implementation paradigm." The Journal of Systems and Software **68**: 183-198.

Ehreke, N. (2005, March 2011). "Lightweight R/O Mapping." from http://onjava.com/pub/a/onjava/2005/12/07/relational-object-mapping.html.

Eisenberg, A. and J. Melton (1999). "SQL: 1999, formerly known as SQL3." SIGMOD Record **28**(1): 119-126.

Eisenberg, A., J. Melton, et al. (2004). "SQL:2003 has been published." SIGMOD Record **33**(1): 119-126.

Fowler, M., D. Rice, et al. (2003). Patterns of Enterprise Application Architecture. Reading, MA., Addison-Wesley.

Fussell, M. L. (1997, 25th September 2007). "Foundations of Object Relational Mapping." Retrieved 25th September 2007, 2007, from http://www.chimu.com/publications/objectRelational/.

Giguette, R. (2006). "Building Objects Out of Plato: Applhing Philisophy, Symbolism, and Analogy to Software Design " Communications of the ACM **49**(10): 66-71.

Grant, E. S., R. Chennamaneni, et al. (2006). Towards analyzing UML class diagram models to object-relational database systems transformations. 24th IASTED international conference on Database and applications Innsbruck, Austria ACTA Press, Anaheim, CA, USA.

Griethuysen, J. J. v., Ed. (1982). Concepts and Terminology for the Conceptual Schema and the Information Base. ISO/TC97/SC5/WG3. New York, ISO.

Hainaut, J.-L. (2006). "The Transformational Approach to Database Engineering." Lecture Notes in Computer Science **4143**: 95-143.

Hall, P., J. Owlett, et al. (1976). Relations and entities. Modelling in Database Management Systems. G. M. Nijssen, North-Holland: 201-220.

Hibernate. Retrieved 11th August 2011, from www.hibernate.org.

Hohenstein, U. (1996). Bridging the Gap between C++ and Relational Databases. European Conference on Object-Oriented Programming, Springer-Verlag, Berlin.

Holder, S., J. Buchan, et al. (2008). "Towards a Metrics Suite for Object-Relational Mappings." COMMUNICATIONS IN COMPUTER AND INFORMATION SCIENCE **8**: 43-54.

Ingres. (January 2006). "Quel Reference Guide." Retrieved 10 September 2011, from http://www.database-books.us/ingres_0016.php.

InterSystems. Retrieved 6th September 2011, from http://www.intersystems.com/cache/.

Ireland, C., D. Bowers, et al. (2009a). A Classification of Object-Relational Impedance Mismatch. The First International Conference on Advances in Databases, Knowledge and Data Applications, Cancun, Mexico, IEEE Computer Society.

Ireland, C., D. Bowers, et al. (2009b). "Understanding Object-Relational Mapping: A Framework Based Approach." International Journal On Advances in Software **2**(2).

Ireland, C., D. Bowers, et al. (2010). Exploring the use of Mixed Abstractions in SQL:1999 - A Framework Based Approach. The Second International Conference on Advances in Databases, Knowledge and Data Applications, Les Menuires, France, IEEE Computer Society.

Ireland, C., D. Bowers, et al. (2011). Exploring the Essence of an Object-Relational Impedance Mismatch - A novel technique based on Equivalence in the context of a Framework. The Third International Conference on Advances in Databases, Knowledge and Data Applications, St. Maarten, The Netherlands Antilles, IEEE Computer Society.

ISO (2003). Part 2: Foundation (SQL/Foundation), ISO/IEC 9075.

ISO6166. Retrieved 14th December 2011, from http://www.iso.org/iso/catalogue_detail?csnumber=33446

JDBC. Retrieved 31st August 2011, from http://download.oracle.com/javase/6/docs/technotes/guides/jdbc/.

JPOX. (2008). "Java Persistent Objects - Object Identity Generators." Retrieved January 2011, from http://www.jpox.org/docs/1_1/identity_generation.html.

Kalman, D. (1994). Moving forward with relational: looking for objects in the relational model, Chris Date finds they were there all the time. DBMS. **7**: 62(6).

Keller, A. M., R. Jensen, et al. (1993). Persistence Software: Bridging Object-Oriented Programming and Relational Databases. ACM SIGMOD international conference on management of data, Washington, D.C, ACM Press.

Keller, W. (1997). Mapping Objects to Tables: A Pattern Language. European Conference on Pattern Languages of Programming Conference (EuroPLoP), Irsee, Germany.

Kent, W. (1991). "A rigorous model of object reference, identity and existence." Journal of Object-Oriented Programming **4**(3): 28-36.

Khoshfian, S. N. and G. P. Copeland (1986). Object Identity. OOPSLA 1986. Portland, Oregon, ACM SIGPLAN: 406-416.

Kochan, S. G. (2008). Programming in Objective-C 2.0, Addison Wesley.

Kotiadis, K. and S. Robinson (2008). CONCEPTUAL MODELLING: KNOWLEDGE ACQUISITION AND MODEL ABSTRACTION. 2008 Winter Simulation Conference, Miami, Florida, IEEE.

Kuhn, T. S. (1970). The Structure of Scientific Revolutions. Chicago, IL, The University of Chicago Press.

Lalonde, W. (1994). Discovering Smalltalk. Redwood City, California, The Benjamin/Cummings Publising Company, Inc.

Lammel, R. and E. Meijer (2006). "Mappings Make Data Processing Go 'Round: An Inter-paradigmatic Mapping Tutorial." Lecture Notes in Computer Science **4143**: 169-218.

Lodhi, F. and M. A. Ghazali (2007). Design of a simple and effectve object-to-relational mapping technique. ACM Symposium on Applied Computing, Seoul, Korea, ACM.

Marcos, E. and J. M. Cavero (2002). Hierarchies in Object Oriented Conceptual Modeling. Advances in Object-Oriented Information Systems : OOIS 2002 Workshops, Montpellier, France, Springer Berlin / Heidelberg

Marcos, E., B. Vela, et al. (2003). "A Methodological Approach for Object-Relational Database Design using UML." Software and Systems Modeling **2**(1): 59-72.

Marguerie, F. (2007). "Choosing an object-relational mapping tool." Retrieved 14th November, 2007, 2007, from http://madgeek.com/Articles/ORMapping/EN/mapping.htm.

Marinescu, F. and R. V. Zicari. (2008). "Java Object Persistence: State of the Union." Retrieved 4th March 2008, 2008, from http://www.infoq.com/articles/java-object-persistence-panel.

Meijer, E. (2006). There is No Impedance Mismatch (Language Integrated Query in Visual Basic 9). OOPSLA, Portland, Oregon, ACM.

Meijer, E. and G. Bierman (2011). "A Co-Relational Model of Data for Large Shared Data Banks." Communications of the ACM **54**(4): 49-58.

Microsoft. (2011). "Microsoft LINQ to SQL." Retrieved 19th September 2011, from http://msdn.microsoft.com/en-us/library/bb425822.aspx.

MicrosoftADO. Retrieved 14th August 2011, from http://support.microsoft.com/kb/173647.

MicrosoftLINQ. Retrieved 14th December 2011, from http://msdn.microsoft.com/en-us/library/bb397910.aspx

MicrosoftODBC. Retrieved 14th August 2011, from http://msdn.microsoft.com/en-us/library/ms710252(v=vs.85).aspx.

Mok, W. Y. and D. P. Paper (2001). On Transformations from UML Models to Object-Relational Databases. 34th International Conference on System Sciences, Hawaii IEEE.

Naiburg, E. J. and R. A. Maksimchuk (2001). UML for Database Design, Addison Wesley.

Neward, T. (2006). "The Vietnam of Computer Science." Retrieved 6th February 2007, from http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx.

Neward, T. (2007, October, 2007). "Avoiding the Quagmire." Retrieved 11th January 2010, from http://odbms.org/vietnam.html.

Niyomthum, K. and S. Chittayasothorn (2003). A Transformation from an Object Database to an Object Relational Database. SoutheastCon, IEEE: 7 - 11.

OMG. "Model Driven Architecture." Retrieved 11th August 2011, from http://www.omg.org/mda/.

OMG. (2004). "Unified Modeling Language (UML) Specification: Infrastructure version 2.0." Retrieved January 2006, 2006, from http://www.omg.org/technology/documents/modeling_spec_catalog.htm.

Oracle. Retrieved 14th August 2011, from http://www.oracle.com/technetwork/java/index-jsp-135919.html.

Peckham, J. and F. Maryanski (1988). "Semantic Data Models." ACM Computing Surveys **20**(3): 153-189.

Petroutsos, E. (2010). Mastering Microsoft Visual Basic 2010, John Wiley & Sons.

Philippi, S. (2005). "Model driven generation and testing of object-relational mappings." Systems and Software **77**: 193-207.

Pizzo, M. (2007). "An Application-Oriented Model for Relational Data." The Architecture Journal(12): 19-25.

Poels, G., A. Maes, et al. (2005). Measuring the Perceived Semantic Quality of Information Models Perspectives in Conceptual Modeling: ER 2005 Workshops CAOIS, BP-UML, CoMoGIS, eCOMO, and QoIS Klagenfurt, Austria, Springer Berlin / Heidel.

Python (2011). Retrieved 14th December 2011, from http://www.python.org

Recker, J. and M. Rosemann (2010). "The measurement of perceived ontological deficiencies of conceptual modeling grammars." Data and Knowledge Engineering **69**: 516-532.

Ruby (2011). Retrieved 14th December 2011, from http://www.ruby-lang.org

Rumbaugh, J., I. Jacobson, et al. (2005). The Unified Modeling Language Reference Manual, Addison Wesley.

Russell, C. (2008). "Bridging the Object-Relational Divide." ACM Queue **6**(3): 18-28.

Sanders, G. L. and S. Seungkyoon (2001). Denormalisation effects in performance of RDBMS. 34th Annual Hawaii International Conference on System Sciences. Hawaii, IEEE: 9.

Schmid, H. A. and J. R. Swenson (1975). On the semantics of the relational data model. Proceedings of the 1975 ACM SIGMOD international conference on Management of data, San Jose, California, ACM Press.

Schwartz, J. and M. Desmond. (2007). "Looking to LINQ." Retrieved 23rd October 2007, 2007, from http://reddevnews.com/features/print.aspx?editorialsid=707.

Smith, J. M. and D. C. P. Smith (1977a). "Database Abstractions: Aggregation." Communications of the ACM **20**(6): 405-413.

Smith, J. M. and D. C. P. Smith (1977b). "Database Abstractions: Aggregation and Generalization." ACM Transactions on Database Systems **2**(2): 105-133.

Smith, J. M. and D. C. P. Smith (1982). Princples of database conceptual design. Database Design Techniques I. S. Yao, S. Nayathe, J. Weldon and T. Kunii, Springer Berlin / Heidelberg. **132:** p114-146.

Soutou, C. (2001). "Modeling relationships in object-relational databases." Data and Knowledge Engineering **36**(1): 79-107.

Stroustrup, B. (1997). The C++ Programming Language, Addison Wesley.

Sybase. Retrieved 14th August 2011, from http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/openclient.

Todd, S. J. P. (1976). "The Peterlee Relational Test Vehicle—a system overview." IBM Systems Journal **15**(4): 285-308.

TopLink. Retrieved 11th August 2011, from http://www.oracle.com/technetwork/middleware/toplink/overview/index.html.

Tsichritzis, D. C. and F. H. Lochovsky (1982). Data Models. New Jersey, Prentice Hall.

Weber, J. (1997). Using Java 1.1, QUE.

WebServices. Retrieved 5 September 2011, from http://www.w3.org/2002/ws/.

Wieringa, R. and W. De Jonge (1991). The identification of objects and roles - Object identifiers revisited, Vrije Universiteit.

Zhang, W. P. and N. Ritter (2001). The Real Benefits of Object-Relational DB-Technology for Object-Oriented Software Development. British National Conference on Databases, Springer-Verlag, Germany.

Zyl, P. V., D. G. Kourie, et al. (2006). Comparing the performance of object databases and ORM tools. Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing couuntries. Somerset West, South Africa, South African Institute for Computer Scientists and Information Technologists.

Appendix A - Definition and Glossary of Terms

Acceptable Solution

An acceptable solution addresses the symptoms of a mismatch rather than the cause.

Appropriate Solution

An appropriate solution addresses the cause of a mismatch.

Conceptual Model

A conceptual model is a description of selected aspects of a universe of discourse without concern for how those aspects may be represented in a computer.

Dialogue

A dialogue at a level of the framework involves an exchange of ideas, across silos of the framework, about a mismatch. A dialogue is concerned with the reason for a mismatch; in other words understanding why it is that data in an object-oriented program will not fit neatly into a relational database.

Equivalence

Two representations are considered to be equivalent, at the schema level of the framework, if they each describe the same semantics of an entity from a universe of discourse. Only those semantics that are equivalent can form part of a non-loss round-trip between an object-oriented program and a relational database.

Implementation Language

An implementation language is a programming language or a language used to describe the schema of a database.

Implementation Model

An implementation model is a representation of a conceptual model as computer software. An implementation model will be described using an implementation language.

Mapping Strategy

A mapping strategy is concerned with a correspondence between those classes in the schema of an object-oriented program and the schema of a relational database.

Mismatch

See Object-Relational Impedance Mismatch

Object-Oriented Program Design

Object-oriented program design is the process of transforming a conceptual object model into the schema of an object-oriented program.

Object-Relational Application

An object-relational application is a computer software system that combines technologies based on the concept of an object and the concept of a relation.

Object-Relational Impedance Mismatch

The term used in this dissertation to refer to a difference between the schema of an object-oriented program and the schema of a relational database.

Paradigm

Whilst the term paradigm was originally intended to describe the set of practices that define a scientific discipline at any particular period of time (Kuhn 1970), it has been used in computing as a label for a particular viewpoint. A particular viewpoint underpins a conceptual framework.

Problem Theme

A problem theme is the label used in this dissertation for a collection of mismatches.

Relational Database Design

Relational database design is the process of transforming a conceptual data model into the schema of a relational database.

Relational Database Schema

A relational database schema is a description of data (Date 1986), p361 and the rules of that data in the language of a particular database.

Round-trip

A round-trip is the term used in this dissertation to describe the transfer of data between an object-oriented program and a relational database. In a round-trip data about an object is stored in a database and, save for its identity, that same object can be re-created in a program at a later time. A round-trip will involve two mapping strategies: one from a program to a database and another from a database to a program.

Schema

A schema is an implementation model, that is, the description of a universe of discourse expressed using a particular implementation language. Specifically a schema is the collection of classes in an object-oriented program that have a requirement for persistence or the description of a relational database.

Silo

A silo is the collection of artefacts representing the separate concerns of an object-oriented program and a relational database across all levels of the framework.

Transformation

When a transformation is applied to a model it produces an equivalent description as another model. The concern of a transformation is orthogonal to that of a mapping strategy. A transformation, such as that from an E-R model to a database schema described using SQL, produces a new model.

Universe of Discourse

A universe of discourse is a model of reality containing all concrete or abstract things that are of interest (Griethuysen 1982), pA-6.

Appendix B - Financial Trading Institution Case Study

The Financial Trading Institution (FTI) is a Bank based in the City of London. Its principal activities are the trading of financial instruments (referred to simply as instruments). Trading involves the purchase and sale of a financial instrument.

The trading activities of FTI are focussed on two kinds of financial instruments: equities and bonds. Specifically the Bank buys and sells equities and bonds on its own behalf (known as proprietary trading) and on the behalf of clients. Ultimately the aim is to make a profit through these dealing activities.

Each financial instrument is assigned a unique ID called the ISIN (International Securities Identifying Number). The structure of an ISIN is defined in ISO 6166 (ISO6166) and is unique across all financial markets.

Equities are also known as shares and the terms are synonymous. The purchase of one share entitles the owner of that share to literally a share in the ownership of a company and potentially the profits.

Equity may be listed on a financial market on any of the many financial exchanges worldwide. A market is a sub-division of an exchange. These sub-divisions are created

by an exchange for many reasons not limited to the value and associated investment risks of the companies listed therein.

A company will apply for a listing of their shares on a market in a specific country because it wishes to raise capital in that country. The process of listing makes a share available to buy or sell on a particular financial market. There are a number of financial regulations that govern the listing of a share on an exchange but if an application is successful a company's equity becomes tradable on a specific market within that exchange.

Bonds are also known as debt instruments. Bonds are a means of financing a company's debt over a period of time (called the term). One way to think about bonds is as a loan from the purchaser of the bond to the issuing company or government.

Rather than owning a share of a company, the purchaser of a bond is buying part of the debt of a company. In so doing the purchaser becomes entitled to interest payments from the company on a bond and the return of their initial investment on maturity if they still own a bond.

Typically a government to finance such things as capital investment issues a government bond. The purchaser of each such bond is effectively lending money to the particular government in question. In return the purchaser is guaranteed the

return of their initial investment at maturity plus interest payments whilst they hold a bond.

Interest payments on a bond can be fixed or variable (also called floating) and are a key source of income from a bond. At the end of the term the holder of a bond also receives a repayment of their initial investment. The holding of a share does not come with such a guarantee. However an issuer of debt can default.

Sales people and traders are just two of the many different kinds of role within the bank. Their role is distinguished from other roles in that they directly earn money for the bank through their work activities.

Sales people and traders are offered an incentive through a profit sharing scheme. The scheme pays them anything from 2% to 10% of profits as an annual bonus subject to achieving targets.

An order is a registration of intent to buy or sell some financial instrument. The trading process at FTI starts either with an order from a client (electronically or by telephone) to buy or sell some financial instrument or from an order raised by one of the FTIs own sales people (a proprietary order).

A sales person can order a quantity of shares in order to develop a position. A position can be long (i.e. a positive number of shares) or short (i.e. a negative number

of shares). A short position can be profitable in a falling market where the price to buy the shares back later will be less than the price at which they were sold.

Holding a position in a stock provides FTI with exposure risk. FTI can choose to take this risk if the research department suggests that the stock is currently undervalued (by holding a long position) or overvalued (by holding a short position).

A trader receives an order to buy or sell an instrument from a Sales person. A trader will deal on a market (i.e. execute a trade) with another trader in another financial institution. A stock exchange will make a charge for executing a trade.

A trade differs from an order in that it represents an actual exchange of financial instruments. Currency is another form of financial instrument so an equity trade is an exchange of a given quantity of shares for a given quantity of currency.

The market capitalisation of a company is the value (price * quantity) of shares in circulation. At FTI, some traders trade in the shares of companies with a small capitalisation (known as a Small Cap. Trader), or large capitalisation (known as a Large Cap. Trader) whilst others (known as Traders) trade in both.

Ultimately the idea is to buy as cheaply as possible and to sell at the highest price possible in order to make a profit. A trader charges a client a commission for a trade to fulfil their order.

A trader can undertake a number of trades in order to complete an order. One reason they do this is to minimize the effect of their trading activity on the market price of the instrument they are trying to buy or sell. If they buy a significant amount of an instrument for example, other traders will identify the significance of this trade and increase the price at which they are willing to sell the instrument. It is hoped that by trading in a large number of small amounts the market price will not be adversely affected.

Proprietary dealing activities are supported by a team of researchers who look for trading opportunities through examination of micro and macro economic factors. The bank can also provide this research information to selected clients for a fee.

At FTI a suite of IT systems that provide a range of services including order entry, trade execution, market connectivity and settlement support trading activity. Technologists provide support for these systems and ensure the continuity of this 24/7 global business.

The business at FTI is the universe of discourse. Figure 32 is a UML class model for the business at FTI. It is only one possible model of the business and it is not the aim of this dissertation to undertake a critique of the model itself. This model is used as the basis of examples throughout this dissertation.

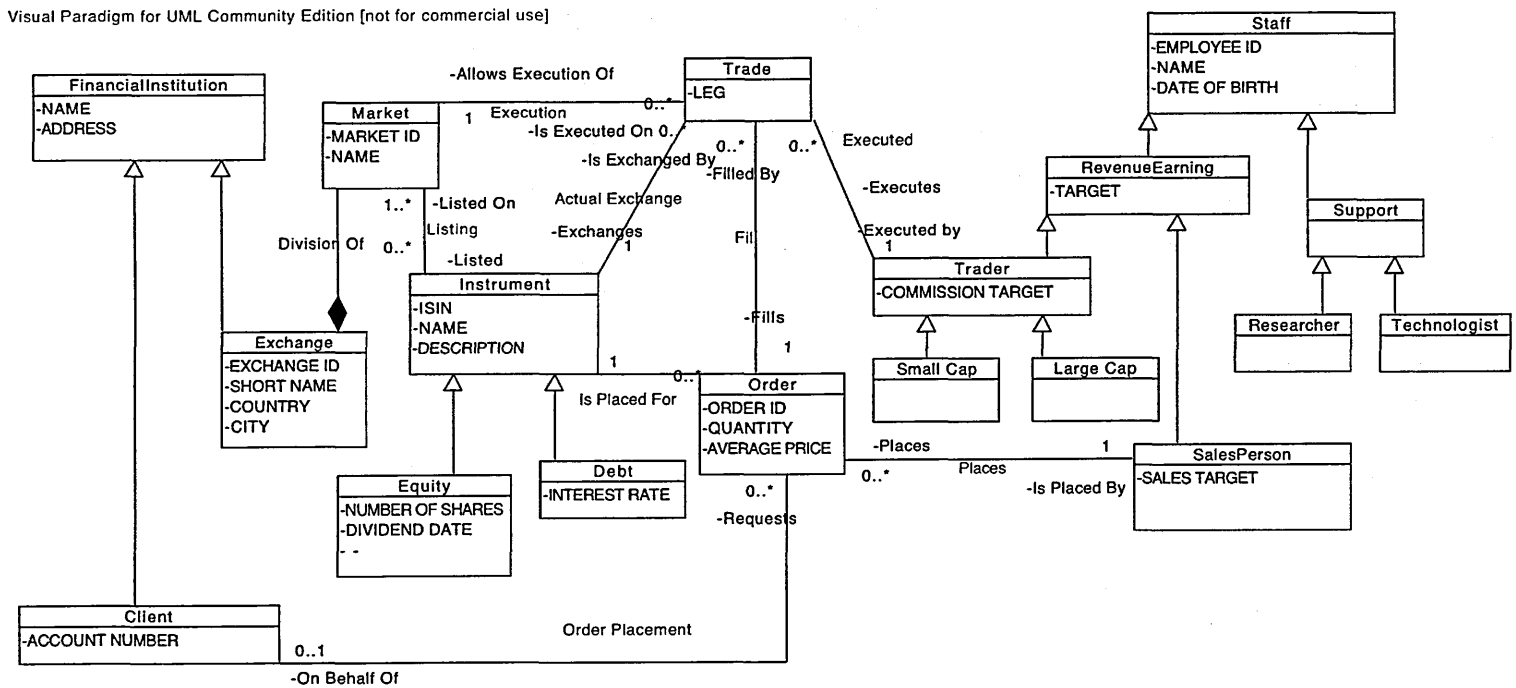


Figure 32 - A UML Class Model of Trading Business at The Financial Trading Institution

Appendix C - The Concept of Identity in an Object-Relational Application

Table 15 - The Concept of Identity in an Object-Relational Application

The Identity of an Object	Issues of Correspondence	The Identity of a Tuple/Row
Concept (Object and Relational) Concern: Whether using a relation or an object the objective is to model some world.		
An object in a computer system is not the same object as that might be observed in the real world. An object is a surrogate of some real-world thing. Object identity is independent of the level of abstraction. For the purposes of identification it does not matter whether an object represents a bolt or a battleship.	Both an object and a relational conceptual framework are concerned with representations of some world. An object and a tuple are distinguished in different ways. An object is unique regardless of abstraction. Whereas an object is unique across all states of the world, an entity key need only be unique in a single state of a relation.	A tuple is distinct by definition (Codd 1970),p379. The identity of a tuple is the value of its constituent domains (identity is based on data value (Tsichritzis and Lochovsky 1982), p52). A representation of identity based on data value relies on an abstraction capturing sufficient information for uniqueness. The identity of a tuple depends on a choice of abstraction. "An identifier [entity] key is some subset of the attributes of an object [entity] which is unique for all objects [entities] in a relation" (Khoshfian and Copeland 1986),p406. An entity key is the representation of an entity identifier in a relation. There are problems with an entity key because the concepts of data value (they are information carrying (Wieringa and De Jonge 1991), p3) and identity are mixed (Khoshfian and Copeland 1986),p406. An entity key has shortcomings (Date 1985),p244: it is subject to change; different key values can identify the same entity; and not all entities may have an entity key.

Language (Java and SQL) Concern: Whether using a class or a table the objective is to model some world **in a computer.**		
In order to represent an object in an information system there must be a way to simulate an object identity (Wieringa and De Jonge 1991),p2. According to Wieringa this is achieved by assigning a globally unique "proper name" or object ID (OID). A proper name cannot change (the persistence principle (Wieringa and De Jonge 1991), p3). There are problems with an OID in the sense described by Wieringa (Wieringa and De Jonge 1991): (1) How to ensure a value is globally unique; and (2) How to determine the first time such an object has been created? If the concept of identity is built into a language then object uniqueness is modelled even though its description is not unique (Khoshfian and Copeland 1986),p406. Addressability (external to an object – a way to access an object) is not the same as identity (internal to an object – a way to represent individuality) (Khoshfian and Copeland 1986), p407 A programming language is typically concerned with addressability and uses this as a basis for an identity. An address is practically unique even though the	Whereas an OID is independent of both data content and structure, an entity key is dependent both on data value and on data structure (Khoshfian and Copeland 1986), p412. It is not certain that an object and an entity key refer to the same entity. Whilst all objects have an OID not all tables must have a primary key. Those tables that do have a primary key should have a different primary key. A choice of primary key is important. Tables with the same primary key might represent data about the same entity.	A SQL table permits two rows with identical values for all columns. Without some basis for distinction there is no way to refer to a particular row. A primary key is a constraint over one or more columns of a table that enforces a uniqueness of value for a row across those columns. A primary key is important because "it is the sole tuple-level [row-level] addressing mechanism within the relational model" (Date 1986),p250. A key value is only unique within a particular table. An entity key or a surrogate key can be a primary key. The value of a primary key is not easily changed if there is a foreign key that references it and that reference is enforced by a foreign key constraint. The use of a surrogate key as a primary key allows an entity key to be changed without restructuring all references.

concepts of an address and an identity are different (Khoshfian and Copeland 1986), p407.		
Schema (the implementation model of an object-relational application) Concern: Whether using a class X or a table Y the objective is to model **a universe of discourse in a computer.**		
The source code for a Java program is a collection of classes. Each class defines the attributes and methods of an object of that class. An object is part of an executing program. A programmer need not be concerned whether an object is unique. An object is unique the moment that it is created. Depending on the design of a program, an object can hold data about an entity or many different entities during the execution of that program. This view is also adopted by the UML. The UML (Rumbaugh, Jacobson et al. 2005),p482 makes a distinction between a snapshot of a system at a point in time in which an object can represent an entity in a model, and an executing system in which an object is a container. Whilst an object is unique, whether the value of an object is, or must be distinct, is a separate question.	The design of both a Java program and a SQL database schema are based on the same universe of discourse. A designer makes a choice of relevant entities from a universe and produces a representation using a particular language. Whilst a choice of an entity must be the same, a choice of abstraction may not. A schema helps to make some assumptions explicit. For example it is possible to begin to see differences in the way each team has represented an entity. It is not certain that an object identity and a surrogate key refer to the same entity.	In an SQL database there is a separation of concerns between data definition and data. A SQL schema comprises a collection of tables, columns and other artefacts that together comprise a representation of a universe of discourse. A relational database is a collection of data that conforms to the rules defined in a SQL schema. It is not always possible to model an entity key so a surrogate key (Date 1985),p244 can be added in order to make a row unique. A surrogate key is a system-generated value. A surrogate key has no meaning in the world being modelled. A surrogate key makes it possible to distinguish things that would otherwise be indistinguishable due to a choice of abstraction. A surrogate key provides value independence (Khoshfian and Copeland 1986), p413 making it possible for the value of an entity key to change. A primary key is semantically meaningful (Chen 1976),p16. The choice of a primary key reflects the semantics of a relation. A surrogate key is generated by a computer system (Date 1985), p245. An entity key or a surrogate key can be used as a primary key. That is a choice of design. An entity key carries information about an entity whereas a surrogate key does not. However, a surrogate key can be promoted to an entity key and so take on a meaning outside the system in which it was devised e.g. a Social Security Number and a Vehicle Identification Number.

		The concept of a primary key forms a cornerstone of normalisation (Codd 1970). Normalisation aims to minimise redundant data. The value of a row must depend on the value of a primary key of that row. However, that which a primary key represents is a choice of design. Using a surrogate key means that none of the data in a row really depends a priori on the value of an entity key.
Instance (an object and a row/column) Concern: Whether using an object or a row the objective is to model **an entity from universe of discourse in a computer.**		
An object is created in order to apply some operation on it. At the moment of creation a Java object is assigned an OID by a Java runtime environment. Kent (Kent 1991),p7 talks about scope. A scope defines the context for an identity. An OID is not "globally unique" (Wieringa and De Jonge 1991), p3. In practice an object need only be unique within the memory space of an executing program. An object exists for the execution of a program so an identity is only in scope during an execution of a program. Two objects can be equal but they are different objects. The content of an object can change without affecting the identity of that object.	An object may not occupy the same address on each execution of a program so the address in memory of an object can change between executions. There are numerous options for creating an OID (e.g. JPOX (JPOX 2008) uses a timestamp or a function to generate a value, whilst Brown (Brown and Whitenack 1994) suggests using a database sequence number generator or column of a row of a table to hold the next available value). So a choice of an option must be made. An object and a row can each represent a container. In this case an object ID and a primary key each provide the identity for a container.	The value of an entity key may be wrong or misrepresented and so subject to change. If a surrogate key is used then an entity key can be changed with little consequence for data integrity in a model. There are benefits in the use of a surrogate key. Kent (Kent 1991),p9 "A computer system expresses an awareness of things, rather than the existence of things". A person can have two social security numbers. A person may need to be recognised by a system but not have a social security number. A surrogate key helps in both these cases. In the first data in two rows can be merged. In the second a row can be created when the value of an entity key is not known.